Performance-based Resource Management in Computational Grid Environment

Thesis submitted for the degree of

DOCTOR OF PHILOSOPHY (Engineering)

of Jadavpur University

October 2007

By

Sarbani Roy

Department of Computer Science & Engineering,
Jadavpur University, Kolkata-700032.

Performance-based Resource Management in Computational Grid Environment

Thesis submitted for the degree of

DOCTOR OF PHILOSOPHY (Engineering)

of Jadavpur University

By

Sarbani Roy

Under the supervision of

Prof. Nandini Mukherjee

Department of Computer Science & Engineering,

Jadavpur University, Kolkata-700032.

October 2007

Certificate from the Supervisor

This is to certify that the thesis, entitled "*Performance-based Resource Management in Computational Grid Environment*", submitted by *Smt. Sarbani Roy*, who got her name registered on 21st January 2004, for the award of Ph.D. (Engineering) degree of Jadavpur University is absolutely based upon her own work under my supervision and that neither her thesis nor any part of the thesis has been submitted for any degree / diploma or any other academic award anywhere before.

Signature of the Supervisor
With date and Office Seal

Dedicated to my beloved parents,

Smt. Irani Ghosh

and

Shri Milan Kumar Ghosh

Acknowledgements

I am deeply indebted to many people who have contributed in making the completion of this dissertation work possible. First and foremost I offer my sincerest gratitude and respect to my supervisor, Prof. Nandini Mukherjee, for her inspiring suggestions, rigorous attention, constant support and monitoring throughout the course of this work. She has constantly forced me to remain focused on achieving my goal. Her observations and comments helped me to establish the overall direction of the research. As my teacher, she has taught me more than I could ever give her tribute here. She has shown me, by her example, what a good mentor and person should be.

I am most grateful to the Rev. Dr. Sailesh Mukhopadhyay, Principal of St. Thomas' College of Engineering and Technology, because he gave me the encouragement to extend my studies to the doctoral level and always keen to provide valuable suggestions. I am also grateful to the friends and colleagues of St. Thomas' College of Engineering and Technology for their support.

I wish to express my gratitude to Prof. Dilip Kumar Basu, Department of Computer Science and Engineering, Jadavpur University, whom I first visited in November 2002, to discuss how I can start my doctoral work. I am most grateful to him because he encouraged me and introduced me to my supervisor, which was a turning point in my life. I also extend my heartiest gratitude to Prof. Mita Nasipuri and Prof. Mahantapas Kundu for their support, encouragement and valuable suggestions and to all my other colleagues in the Jadavpur University for their cooperation in several ways.

I express my sincere thanks to the Prof. Atal Chowdhury, Head of the Department, Department of Computer Science and Engineering, Jadavpur University, for allowing me to use the Alpha-lab and DST-FIST lab. I am also thankful to the co-ordinators of CMCC, Jadavpur University, for providing the access to IBM Power4 pSeries 690 Regatta server.

I would like to express my sincere thanks to the professors and research scholars of Technical University of Munich, Germany and other group members of Jadavpur University of DST-DAAD project (2004-2006) entitled “Performance Monitoring and Analysis of Large Distributed Systems using Mobile Agents” for generously sharing their views and knowledge in our collaborative project.

In my daily work I have been blessed with a friendly and cheerful group of fellow research scholars and students. In particular, I would like to thank Sushanta, Saikat, Subhashis, Sohini, Monideepa, Ajanta, Ruben, Madhulina, Madhumita for their cooperation and useful discussions I had with them during my research work. Special thanks to Swarnajyoti for being a friend and inspired me by his dedication towards his work.

I am indebted to my parents who through their perseverance, inspiration and untiring effort helped me to reach the place I acquire now. My heartfelt gratitude is extended to my husband Sujit, for his incessant moral support during the tenure of my dissertation work. I would also like to express my heartfelt gratitude to my in-laws and other family members for their constant support and encouragement. Last, but not the least I thank to all my teachers, friends and relatives for their good wishes and support.

Jadavpur University. Sarbani Roy

October 2007.

Abstract

Tasks of a resource management system are complicated in a heterogeneous, dynamic environment like Grid. A resource management system may have to deal with the resources on which it has no control and which are under the control of multiple administrative domains. Moreover, the system loads and status of the resources change frequently. So the information about the resources is often limited or dated. In particular, in a computational Grid environment, the role of a resource management system becomes important, as the quality of services required for the computational jobs (in terms of execution time or performance) must be maintained.

This thesis proposes an integrated framework for performance-based resource management in computational Grid environment. Objective of this thesis is to design and implement this framework for pursuing two definite goals – maintaining the quality of services required for a batch of jobs (possibly components of a single application), while keeping the resource utilization cost as minimum as possible. A multi-agent system (MAS) is developed using a software engineering approach based on Gaia methodology in order to support this framework. It implements different resource brokering strategies. These strategies help in deciding overall optimal usage of resources for executing multiple concurrent jobs in a Grid environment. The MAS provides adaptive execution facility either by rescheduling the jobs onto different resource providers or applying local tuning techniques to jobs in case of any performance problem (may be due to some change in the resource availability or usage scenario), thereby, maintaining the quality of service (desired by the client). The thesis demonstrates the effectiveness of the multi-agent system for managing execution of multiple concurrent jobs by developing a tool, called PRAGMA (**P**erformance based **R**esource Brokering and **A**daptive execution in **G**rid for **M**ultiple concurrent **A**pplications). This work also evaluates the efficiency of this tool by performing different sets of experiments using different batches of jobs (benchmark codes) on a local Grid test bed.

List of Publications of the Author Related to the Thesis

[1] Sarbani Roy and Nandini Mukherjee – "A Study on Performance Monitoring Techniques for Traditional Parallel systems and Grid", In *Proceedings of IFIP International Conference on Network and Parallel Computing (NPC 2004)*, LNCS, vol. 3222/2004, pp. 38-46, Wuhan, China, October 18-20,2004.

[2] Sarbani Roy, Subhasis Dasgupta and Nandini Mukherjee – "Performance Tuning in Grid Environment", In *Proceedings of National Conference on Networks and Distributed systems (NETWORKS 2005)*, CSI Conference (Div V, Div VII and Hyderabad Chapter), pp. 8-15, Hyderabad, India, February 25-26, 2005.

[3] Sarbani Roy, Saikat Halder and Nandini Mukherjee – "A Multi-agent Framework for Performance Tuning in Distributed Environment", In *Web Proceedings of International Conference On High Performance Computing (HiPC 2005)*, Goa, India, December 18-21, 2005.

[4] Sarbani Roy and Nandini Mukherjee – "Towards an Agent-Based Framework for Monitoring and Tuning Application Performance in Grid Environment", In *Proceedings of 2nd International Conference on Distributed Computing & Internet Technology (ICDCIT 2005),* LNCS, vol. 3816/2005, pp. 229-234, Bhubaneswar, India, December 22-24, 2005.

[5] Sarbani Roy and Nandini Mukherjee – "Utilizing Jini Features to Implement a Multiagent Framework for Performance-based Resource Allocation in Grid Environment", In *Proceedings of International Conference on Grid Computing and Applications (GCA 2006).* The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing, pp. 52-60, Las Vegas, June 26-29, 2006.

[6] Sarbani Roy, Sohini Dasgupta and Nandini Mukherjee – "A Multi-agent Framework for Resource Brokering of Multiple Concurrent Jobs in Grid Environment", In *Proceedings of - International Symposium on Parallel and Distributed Computing (ISPDC 2006)*, pp. 329-336, Timisoara, Romania, July 6-9,2006.

[7] Ajanta De Sarkar, Sarbani Roy, Sudipto Biswas and Nandini Mukherjee – "An Integrated Framework for Performance Analysis and Tuning in Grid Environment ", In *Web Proceedings of International Conference On High Performance Computing (HiPC 2006)*, Bangalore, India, December 18-21, 2006.

[8] Madhulina Sarkar and Sarbani Roy- "Grid Portal for Optimal Resource Brokering within Multi-agent Framework", Poster presentation in *IEEE WIE National Symposium on Emerging Technologies (WieNSET 2007),* pp. 6, Kolkata, India, June 29-30, 2007.

[9] Sarbani Roy, Madhulina Sarkar and Nandini Mukherjee – "Optimizing Resource Allocation for Multiple Concurrent Jobs in Grid Environment", Accepted for publication, In *Proceedings of Third International Workshop on scheduling and Resource Management for Parallel and Distributed systems (SRMPDS 2007),* Hsinchu, Taiwan, December 5-7, 2007.

[10] Sarbani Roy, Madhulina Sarkar and Nandini Mukherjee- "Implementation of a Resource Broker for Efficient Resource Management in Grid Environment", Accepted for publication, In *Proceedings of 15th International Conference on Advanced Computing & Communication (ADCOM 2007),* Guwahati, India, December 18-21, 2007.

List of Presentations in National / International Conferences

[1] "Performance Tuning in Grid Environment", presented in *National Conference on Networks and Distributed systems (NETWORKS 2005)*, CSI Conference (Div V, Div VII and Hyderabad Chapter), Hyderabad, India, February 25-26, 2005.

[2] "An Integrated Framework for Performance Analysis and Tuning in Grid Environment", presented in *International Conference On High Performance Computing (HiPC 2006)*, Bangalore, India, December 18-21, 2006.

[3] "Grid Portal for Optimal Resource Brokering within Multi-agent Framework", presented in *IEEE WIE National Symposium on Emerging Technologies (WieNSET 2007),* Kolkata, India, June 29-30, 2007.

Contents

List of Figures

List of Tables

Glossary

The terminologies, which are defined and/or used in this thesis, are as follows:

Multi-Agent System (MAS): Within a multi-agent system many intelligent agents interact with each other.

JDesc: Description of a job given by the client.

Job Requirement List (JRL): Job Requirement List identifies the minimal requirement to run a job.

ResourceSpecificationMemo: ResourceSpecificationMemo specifies the resources owned by a particular resource provider that are available for use by the Grid clients.

ResourceSpecificationTable (RST): A ResourceSpecificationTable is a collection of several ResourceSpecificationMemo.

ResourceBroker (RB): ResourceBroker bears the responsibility of finding suitable resources for multiple jobs submitted to the system by one or more clients.

ResourceProvider (RP): ResourceProvider at a particular Grid Site prepares a ResourceSpecificationMemo specifying the resources that are available for the clients.

JobController (JC): JobController is responsible for controlling the execution of all the concurrent jobs (components or tasks of an application).

JobExecutionManager (JEM): JobExecutionManager becomes associated with each job and controls and keeps track of its execution.

Service Level Agreement (SLA): Service Level Agreements provide a mechanism to match the clients' requirements with the capability and the status of resources offered by the resource providers.

ResourceProviderList: ResourceProviderList of a job is a collection of resource providers, which are capable and ready to provide computational services to that job.

JobResourceMatrix: JobResourceMatrix is the collection of the ResourceProviderLists for all the jobs in a batch.

JobMap: Allocation of a job to the selected resource provider.

CostMatrix: Entries in the CostMatrix are costs, which are used to decide an optimum resource allocation strategy.

CostVector: CostVector for a job is the corresponding row in the CostMatrix.

PerfData: Performance data regarding execution of a job and the corresponding environment.

AnalysisReport: Analysis report is prepared after analyzing performance data.

Expected Completion Time: Expected completion time of a job specified in the JRL and SLA (as estimated and desired by the client).

Estimated Completion Time: Estimated completion time of a job computed by the analysis agent.

Chapter 1

Introduction

The modern era of computer science is dominated by high-speed machines with incredible processing capabilities, complex data storage methods, new-generation operating systems, and advanced networking services. However, in spite of this tremendous technological advancement, everyday, computer scientists face new and diverse challenges from the users in industries and businesses because of their increasing requirements for computational power, storage capacity, and high network bandwidth. With a vision of fulfilling the resource demands of these users in all aspects, computer scientists are looking towards *harnessing the power of the complex and powerful, but geographically dispersed computing resources in the form of a grid*. The concept has been primarily inspired by the conventional utility power grids, which provide power to our homes and businesses everyday.

This chapter focuses on the unique features of Grid and its application in the context of high performance computing. It also highlights the fact that resource management in Grid involves non-trivial operations and summarizes its needs and requirements. The thesis is dedicated to developing an efficient resource management system for Grid and this chapter explains the motivation and major contributions made in this thesis. At the end, it presents an overview of organization of the thesis.

1.1 Grid Computing

The term *Grid Computing* refers to efficient utilization of the power of a potentially unlimited number of ubiquitous computing devices connected within a grid. Grid computing has its genesis in distributed computing. In fact, Grid computing may be thought as one step ahead of it. Distributed computing began in the early 1970s, when computers were first linked by networks. It proliferated with the emergence of two technologies. First, the microprocessor technology has made computers commodity items in the form of minicomputers, then workstations, and then personal computers. Alongside the world also observed enormous advancement in computer networking technologies. Eventually, in 1990s, distributed computing boosted up with the maturity of the Internet and the notion of Grid was introduced as the advanced stage of application of the distributed computing technologies with an objective of creating a more generic resource sharing context.

The idea of Grid has been conceived as a large scale, generalized *distributed network computing* system that can scale to Internet-size environments with machines distributed across multiple organizations and administrative domains. In our daily life, a person can walk into a room and turn on the lights without knowing from where exactly the power is drawn to the intended devices at that moment of time. In the same fashion, just like the electricity and water supply services, which are available on demand, Grid researchers envisage to make the computational power (and other such services) to be available *on demand*. Furthermore, Grid computing seeks for (and is capable of) adding a large number of computing, storage and other devices *dynamically* into any Grid environment, thereby adding to the capacity of the environment as and when required.

The worldwide Grid communities engage themselves in continuous development of related services and standards, so that the complex problems of the contemporary industries and the business world that cannot be solved using traditional, single computational platforms (even with largest power), can be ported and solved in a Grid environment. However, with all the benefits of Grid computing, as discussed in details in [28, 29], it must be recognized

that a Grid is a complicated global environment, which aims at supporting a multitude of open standards and technologies in a wide variety of implementation schemes. Thus, further concentrated research efforts are required to make the dream of the computer scientists a success.

1.2 High Performance Computing and Grid

One objective of Grid computing is to aggregate and utilize the computational power of ensembles of shared, heterogeneous, and distributed resources (potentially controlled by separate organizations) for executing compute-intensive applications of the users who do not have sufficient resources (in terms of quality or quantity) under their own control. Such an environment is termed as a *computational Grid environment*. The idea of harnessing underutilized computational power of the resources on different nodes had been introduced in the early days of distributed computing. In fact, the first full-fledged distributed computing effort was a piece of software, which could move from one computer to another and get executed using its idle resources for the benefits of the users. The primary advantage of Grid environment is that the combined capacity of many high and low-powered nodes (each node may be a personal computer, a workstation, a cluster or a large powerful computer) can provide computational power similar to a huge multiprocessor supercomputer, but at a lower cost. This is predominantly due to the economies of scale of producing commodity hardware compared to the lower efficiency of designing and constructing a small number of custom supercomputers. Foster and Kesselman [29] describes a computational Grid as an environment which consists of one or more hardware- and software-enabled environments for providing *dependable, consistent, pervasive and inexpensive* access to high-end computational capabilities.

The primary performance disadvantage of Grid environment is that the processors and local storages on various nodes are connected to a network with comparatively low-speed communication capabilities. Thus, unlike traditional high performance computing environments (HPC systems including multiprocessor supercomputers), this arrangement is well suited to applications in which multiple parallel computations can be executed

independently, without the need to communicate intermediate results between the processors. Within a traditional HPC environment, the way an application is divided is strictly controlled. On the contrary, Grid uses a more flexible model, which allows an application to be divided into work units of nonstandard-sizes that can be distributed among disparate Grid nodes. The execution of an application is managed by executing these work units in parallel. The Grid thus becomes a generalized computing resource, with the work being distributed among the individual nodes and running until the jobs and operations are completed. However, if the work units are dependent and require communications among themselves, Grid may not be suitable as an HPC environment.

There are other differences between a traditional HPC environment and a Grid environment. Most of the traditional HPC platforms are based on fixed and dedicated hardware, specialized operating systems and software that work together to produce a high performance environment. On the other hand, specialized and dedicated components are not necessary in building a Grid environment. Grid can be expanded with further nodes more easily in comparison with a typical HPC environment. This high-end scalability of geographically dispersed Grid nodes is generally favorable, when it is combined with the low need of connectivity between nodes with respect to the capacity of the public Internet. Conventional HPC systems also pose physical challenges in supplying sufficient electricity and cooling capacity at a single location.

1.3 Resource Management

It has already been pointed out that a Grid platform is based on a complex distributed architecture characterized by the heterogeneity of its resources. Moreover, Grid provides a highly dynamic environment in which resources/services can be added or withdrawn at any point of time. Managing such dynamic, heterogeneous pool of resources and allocating the resources to Grid applications in the most effective manner is one of the central tasks in a Grid environment. Thus, *resource management* is one of the focus areas in Grid computing research.

"The term *resource management* refers to the operations used to control how capabilities provided by Grid resources and services are made available to other entities, whether users, applications, or services" [23]. Resource management in Grid involves various issues. The basic responsibility of any resource management system is to accept the client's request for resources for the execution of a job and then find and allocate a suitable resource provider for that job. Resource managers in traditional computing environments are local, have complete control of a resource and can implement the mechanisms, policies etc. for that resource separately. On the other hand, a resource management system in Grid environment manages resources that span across multiple administrative domains, and also deals with the problems due to heterogeneity and dynamicity in the environment. Some of the early works have concentrated on tackling heterogeneity in such environments [6, 20, 63]. However, most of the recent works [26, 75] focus on dealing with the facts that different organizations operate their resources under different policies, and the goals of the resource user and the resource provider may be different and even conflicting.

One major objective of resource management in a computational Grid environment is to allocate jobs to make efficient use of the computational resources under different resource providers and, thereby, achieve high performance of the jobs. Therefore, performance of these jobs must be regularly monitored and considered by the Grid resource management system. In a traditional computing environment, performance of a particular job executing on a single resource is taken into account. However, in a Grid environment, when different work units (components) of a single application execute on different resources, overall performance of all these components must be considered. Alongside, status of the underlying resources (such as availability and current usage) should also be taken into account so that none of these components suffer.

The components of an application may be of different types and may have different resource requirements. For high performance, the requirements of maximum possible components of an application must be fulfilled at a time. In addition, these components must adapt to the environment at any point of time in order to guarantee the specified level

of performance (expressed as the *Quality of Service* (QoS)) in spite of the variety of run-time events that cause changes in the availability of resources. In such a scenario, a resource management system should have the ability to discover, allocate, and negotiate the use of resource capabilities, not only at the time of submitting the components of an application, but also at the time of executing them on the basis of their measured performances and the resource usage scenarios.

The above discussion reveals that Grid resource management is not just about scheduling jobs on the fastest machines, but rather about scheduling all computational jobs and all data objects on machines whose capabilities match the requirements, while preserving site autonomy, recognizing usage policies and respecting conditions for use. This thesis centers around the issues related to *performance-based resource management* in Grid environment and focuses on designing and implementing novel strategies in order to tackle these issues.

1.4 Motivation

The previous section emphasizes the importance and highlights the requirements of a Grid resource management system. There are many ongoing Grid computing projects, which particularly focus on various topics related to resource management in Grid. However, most of them [6, 21, 63] follow a system-centric approach and concentrate on the activities like resource discovery and scheduling onto resources based on resource availability, usage policies, current loads on resources etc. Only a few [44, 75] support resource allocation and application scheduling algorithms, which are driven by the users' quality of service requirements. In particular, there is hardly any focus on developing efficient software tools for resource management in computational Grid environment (except few like ICENI and GrADS projects [37, 44]) keeping in mind the critical performance criteria of the jobs.

This thesis argues that an integrated approach towards performance-based resource management is one of the necessities for high performance computing in Grid. The clients of computational services in Grid environment need to specify their requirements and the resource management system must be able to match these requirements with the dynamic

resource availability and changing resource usage scenarios. Moreover, if the resource availability and the usage scenario change during runtime, the system needs to adapt to these changes.

As discussed earlier an application may be broken into multiple concurrent jobs (work units having no communication between them), which are executed parallely on different resources in order to reduce the overall execution time. The jobs may have different resource requirements and should be allocated onto the resources accordingly. Thus, *coordinated management* of multiple resources is an important issue that needs to be addressed. It has to be seen that the requirement of every job is fulfilled, while no over-provisioning is made and thus the resource usage is optimized.

In a heterogeneous distributed system like Grid, a centralized resource management system may become a bottleneck. On the other hand, a distributed resource management system may not have a global view of the environment and therefore, may not be able to take appropriate decisions involving different components, resources and organizations. Hence, a hybrid approach is needed. Also the components of a resource management system must be interoperable, scalable and flexible. As the thesis will demonstrate, mobility feature of these components may also be useful.

Keeping the above requirements for a Grid resource management system in mind, the contributions made by this thesis are summarized in the next section.

1.5 Contribution

This thesis aims at providing an integrated framework for performance-based resource management in Grid environment. The distinguishing features of this framework in comparison with other systems are:

- It supports concurrent execution of multiple components of an application on different Grid resources in order to achieve high performance.

- It creates agreements between the resource providers and resource consumers and always abides by these agreements.
- It attempts to meet up the resource requirements for every component of an application (every job), but also sees that no overprovisioning is made.
- The resource allocation to the components (jobs) is made in an adaptive manner and any change in the environment is taken care of.

A major part of the work relies on software agent technology. Software agents can facilitate transparent and robust distributed communications. As a result, software agent technology provides capabilities that are perfectly associated with the needs of a Grid resource management system as laid down in this thesis. Hence, a significant contribution of this thesis is a principled design of a multi-agent system based on the proposed resource management framework.

An application, which is essentially a collection of independent concurrently executable jobs (the term "job" refers to the work unit) with no communication requirement, i.e., not sharing any workflow or dependences is submitted by the client to the multi-agent system. The collection of these jobs form a batch. The initial task of the multi-agent system is to assign each job in the batch to an appropriate resource provider so that the selected resource providers fulfill all the requirements of the jobs. At the same time the selection must be a cost effective choice for that job. Efficient algorithms and strategies that take care of these issues are proposed in this thesis. In case the resource provider is unable to fulfill the requirements of the job (due to the changes in availability or loads on that particular resource provider), it needs to be reallocated to a different resource provider. A mobile agent-based framework is also proposed and implemented in this thesis that checkpoints and migrates the suffered job to another resource provider and thereby maintains the desired performance.

The contributions made in this thesis are summarized as below:

- The thesis proposes an integrated framework for managing resources for execution of a batch of jobs in a Grid environment for high performance.
- To support this framework, a multi-agent system is designed and developed using a software engineering approach based on Gaia methodology [117, 120]. The major characteristics of this system are:
    - Within the system, autonomous agents cooperate with each other in order to fulfill the goals set out for the system, i.e. to abide by the agreements between the resource providers and resource consumers.
    - The multi-agent system adopts a combination of both central and distributed approaches for resource management.
    - The agents within the system offer services for resource brokering, performance analysis, local tuning and rescheduling of jobs depending on the changing resource requirements or usage scenarios.
- Novel approaches to resource brokering are suggested and implemented to support a cost-effective allocation of jobs onto resources.
- A mobile agent framework is developed within which a set of mobile agents is deployed to support scheduling and rescheduling of jobs onto the resources. These agents are also part of the multi-agent system and cooperate and collaborate with other agents (which are generally static) in order to increase the efficiency of the system.
- Simple local tuning techniques (based on performance analysis of the jobs and the environment) are suggested in order to adapt with any changes in the environment. Although much detailed study on the tuning techniques has not been brought within the scope of this thesis.
- A tool, PRAGMA (**P**erformance based **R**esource Brokering and **A**daptive execution in **G**rid for **M**ultiple concurrent **A**pplications) is developed to evaluate the effectiveness of the multi-agent system and to test the algorithms suggested as part of this work. PRAGMA is deployed on top of the Globus Toolkit 4 [41], which is the most used Grid middleware.

PRAGMA is implemented in Java and designed to support the resource brokering and adaptive execution of computational jobs in Java and C. PRAGMA provides services for both sequential and parallel jobs. OpenMP [78] programming model is used for parallel jobs written in C and JOMP (Java OpenMP) [9] parallel programming model is used for parallel jobs written in Java.

1.6 Organization of the Thesis

The chapters of this thesis are organized as follows. An overview of the Grid environment and its middleware is presented in *Chapter 2*. In particular, this chapter makes a thorough study of the most used Grid middleware, the Globus Toolkit [41]. *Chapter 3* discusses resource management techniques used in the existing systems and proposes a framework for performance-based resource management in Grid. This chapter also outlines the objective of the framework. *Chapter 4* presents the detailed design of the Multi-Agent System (MAS) and gives an overview of PRAGMA. This chapter briefly highlights the ingredients of a software engineering approach, namely Gaia that has been used in this thesis for developing the proposed MAS. Different agents and their interactions are described using some abstract and concrete models based on Gaia. *Chapter 5* describes how the multi-agent framework facilitates resource brokering in Grid environment and also discusses the implementation details of resource brokering in PRAGMA. The effectiveness of PRAGMA in resource brokering is evaluated in *Chapter 6*. Applicability of various techniques for monitoring the applications and resources in Grid environment is briefly discussed in *Chapter 7*. Performance analysis is a well-studied problem in high performance systems and many researchers are currently working for developing novel techniques [39, 72, 85, 90, 104, 107, 114] for performance analysis in Grid systems. This thesis does not endeavor to get into the depth of performance analysis problems in Grid. However, case studies on representative performance monitoring and analysis tools for traditional parallel systems and Grid are furnished in this chapter. It also highlights the main issues that need to be tackled while designing a Grid monitoring system. Next two chapters, deal with the adaptive execution facility of the multi-agent framework. Implementation of this framework using Jini mobile agents is described in *Chapter 8*. This

chapter also explains the implementation details of PRAGMA in order to fulfill this feature. *Chapter 9* presents the experimental results of evaluation of this feature in PRAGMA. *Chapter 10* concludes with summaries of the work presented in the thesis, its limitations and suggestions of its extension that can be carried out in future.

Chapter 2

Grid Computing – Overview and Concepts

"*A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities*." – Ian Foster and Carl Kesselman, The Grid: Blueprint for a Future Computing Infrastructure. [29]"

During the last two decades, an evolution in the Internet technology has occurred. Alongside, powerful computers and high-speed network technologies have become available as low-cost commodity components and combination of these two has changed the way we use computers today. These technology opportunities have led to the possibility of using distributed computers as a single, unified computing resource, leading to what is popularly known as Grid computing [28, 29]. The term Grid is chosen as an analogy to a power grid that provides consistent, pervasive, dependable, transparent access to electricity irrespective of its source. It provides an execution environment in which high speed networks are used to connect supercomputers, clusters of workstations, databases, scientific instruments located at geographically distributed sites and owned by different organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce [18]. Other similar approaches to network computing also evolved, such as metacomputing, scalable computing, global computing, Internet computing, and more recently peer-to-peer (P2P) computing [10, 80].

An overview of Grid environment is presented in this chapter. The chapter also highlights how Grid is different from other distributed systems. The concepts related to Grid middleware is introduced which is followed by a discussion on the Globus Toolkit – an established Grid middleware among the scientific community. Different components of Globus Toolkit and their working procedures are also discussed in this chapter.

2.1 Overview of the Grid Environment

The concept of Grid has evolved from the idea of fulfilling the resource requirement of an application with the help of coordinated resource sharing among dynamic collections of individuals, institutions, and resources. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully what is shared, who is allowed to share, and the condition under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form a group called virtual organization (VO) [30, 58]. Thus, creating virtual organizations and enterprises as a temporary alliance of enterprises or organizations that come together to share resources and skills, core competencies, or resources can be enhanced (multiplied) in order to better respond to business opportunities or to fulfill large-scale application processing requirements as envisioned in [15]. The cooperation among the VOs is supported by computer networks.

The concept of Grid computing started as a project to link geographically dispersed supercomputers, but now it has grown far beyond its original intent. The Grid infrastructure can benefit many applications, including collaborative engineering, data exploration, high-throughput computing, and distributed supercomputing. The environment can be better visualized as a set of computer resources spread all over the world, belonging to one or many organizations (private or public) that are shared by a user community under specific constraints. The Grid concept arises from the following considerations:

- Many CPU resources remain unused for variable amounts of time.
- Modern research works and projects require intensive computations.
- Modern PCs offer very high-level performances at low cost.

- Nearly every computer is connected to Internet and has access to all other computers in the net.

Therefore interconnecting a large number of computers (virtually all the computers in the world) that run specialized managing software, a Grid can be setup and an extremely powerful virtual computing environment can be created where every institution and individual may subscribe and run complex applications. Some of the advantages of Grid computing that have been envisaged by the researchers [29] are discussed below:

- An existing application can be run on different machines, thereby efficiently utilizing underutilized resources. For example, the machine on which the application is currently running might be overloaded with several activities. The application in question could be moved and run on an idle remote machine elsewhere on the Grid. The remote machine must meet all resource requirements required by the application.
- The potential for massive parallel computing utilizing many distributed computational resources is one of the most attractive features of a Grid. A compute intensive Grid application can be decomposed into many smaller sub-tasks, each executing on a different resource.
- An application is benefited from a Grid environment when the resource requirement cannot be fulfilled (either quantitatively [1] or qualitatively [2]) from the resources owned by the user. Thus, a Grid provides a good basis of creating a collaborative environment.
- A Grid can also provide access to additional resources, in addition to processors and storage resources, such as library utilities.
- Resource balancing is another important feature available in Grid computing. For example, an unexpected peak can be routed to relatively idle machines in the Grid.
- It is possible to enable and simplify collaboration among a wider audience. Grids are persistent environments that enable applications to integrate computational and information resources, instruments and other types of resources managed by diverse

[1] to provide more resources than available locally. Thus, resources may be unified to speed the execution or to execute significantly larger and more complex problems.

[2] to provide more special resources than available locally. Performance is linked with the resource quality and availability.

organizations in widespread locations. The participating resources of the Grid can be organized dynamically into virtual organizations. These virtual organizations can share their resources collectively as a larger Grid.

Grid can be envisaged as a step forward than distributed systems. Next section provides a formal view of Grid computing and the following section discusses how Grid is different from distributed systems and focuses on the issues and challenges that need to be dealt with by a Grid middleware.

2.2 Formal View of Grid Computing

Parashar et al presents a formal view of the concept of Grid computing in [88]. As discussed in [88], the idea of Grid computing builds on conceptual and implementation models. Abstract machines for programming models and systems to enable application development in Grid are defined by conceptual models. Implementation models deal with the virtualization of organizations, the creation and management of virtual organizations, and the instantiation of virtual machines as the execution environment for an application as shown in Figure 2.1. Resources are organized in a Grid environment among heterogeneous distributed dynamic virtual organizations (VOs). Virtual organizations composed from resources provided by multiple concrete organizations. A virtual organization (VO) can be formally defined as a tuple (O, RS, I, PY, PL) with the following meanings:

O→ the set of concrete organizations ({o}) forming an instance of VO.

RS→ the set of resources and services ({rs}) supported by VO.

I→ the interface for accessing RS.

PY→ the set of policies ({py}) for the operation of VO.

PL→ the set of protocols ({p}) for the implementation of PY.

A concrete organization (o) can be defined as a tuple (RS, I, PY, PL) where RS, I, PY and PL are used with the same meaning as described above. The main objective of a virtual organization is to provide facilities that are not feasible within the capabilities of a single concrete organization. Each concrete organization within the VO may both contribute

resources to the VO and use resources provided through the VO. A virtual machine (VM) can be defined as a tuple (VO, RS, OP, SR), which is a composition of resources within a virtual organization and defines an instance of the execution environment of an application with the following meanings:

RS→ the set of resources belonging to the VO from which the VM is composed.

OP→ the set of operations ({op}) executed by a VM.

SR→ the set of sequencing rules ({sr}) which constrain ordering and composition of operations.

A virtual machine is implemented using the set of middleware services ({ms}) made available through its containing VO.

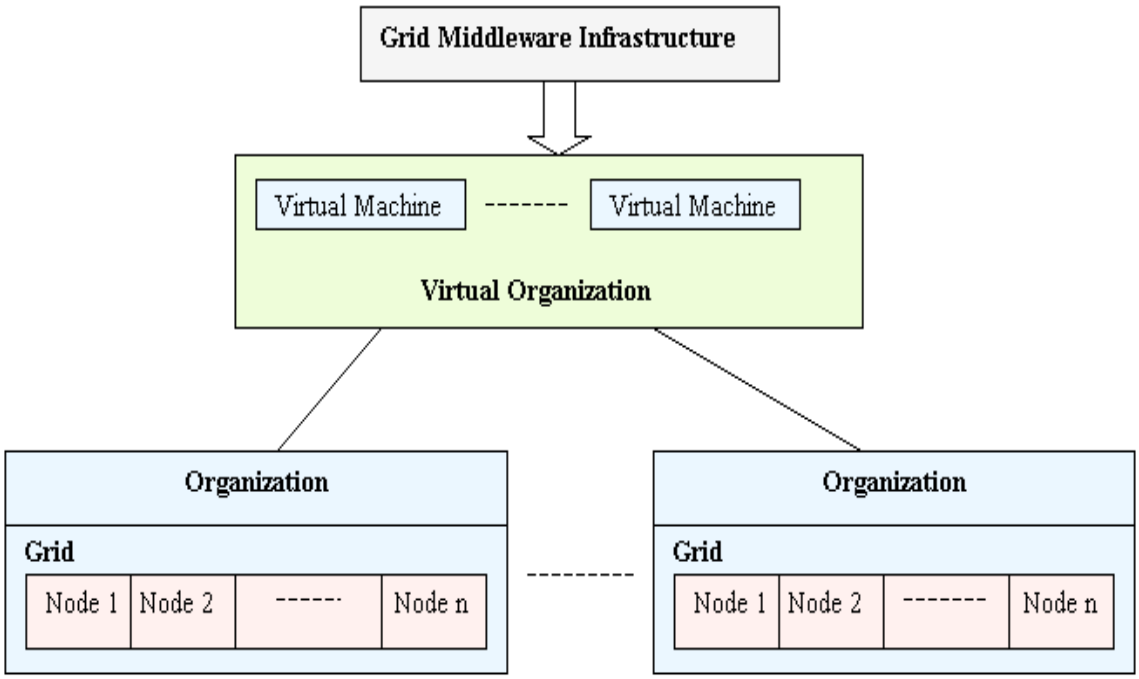


Figure 2.1: Implementation model for Grid computing.

2.3 Distributed Systems versus Grid

Grids are usually viewed as the successors of distributed computing environments; although from the user's point of view conventional distributed systems and Grids are essentially different. Nemeth, in [71], distinguishes between distributed systems and Grids. A conventional distributed environment assumes a pool of computational nodes, whereas, Grid assumes a virtual pool of resources rather than computational nodes. Although, many of the Grid environments are currently focusing on only computational resources (each consisting of CPU cycles and memory) [29, 71], the ultimate objective is to incorporate other resources like storage, network, data, software and sensors. Computational resources used in a Grid can be PCs, workstations or supercomputers. These resources are not only geographically distributed; they span multiple administrative domains and adopt various security policies.

The concept of Grid is often mixed up with the concept of cluster. A computer cluster is a distributed system that consists of a group of tightly coupled computers that works together closely to behave like a single, unified computing resource. The key differences between Grids and traditional clusters are that Grids connect collections of computers, which do not fully trust each other, and hence operate more like a computing utility than like a single computer. Unlike Grid environments where parallel jobs can be executed in a distributed manner using resources managed by multiple administrators, a cluster is usually supervised within a single administrative domain. Besides, Grids typically support more heterogeneous collections than are commonly supported in clusters [64].

The use of the Grid resources is usually characterized by their availability outside the context of the local administrative domain [30]. This *external provisioning* approach requires creating a new administrative domain, which is referred as a Virtual Organization (VO) (see Section 2.2) with a distinct and separate set of administrative policies (home administration policies together with external resource administrative policies form the VO administrative policies). When users from multiple organizations require to make use of a

computational Grid efficiently and securely, they must belong to one virtual organization (VO) sharing a common goal and a common set of resources.

In distributed systems, the virtual pool of nodes is static. Hence, while running an application, all processes are mapped to nodes chosen from pool either by the user or by the application itself or by some kind of scheduling mechanism. This is possible because the user has the knowledge of the capabilities and the features of the nodes. Distributed systems first find an appropriate node for the application and then satisfy the resource needs locally. Nodes at the virtual level in a distributed environment are directly mapped onto the physical computational nodes. The virtual layer is just a different view of the physical layer and not really a different level of abstraction. Nodes appear on the virtual level exactly as they are at physical level with the same names and capabilities. An authorized user of a node (realized by valid login to that node) can access all resources belonging to that node.

On the contrary, a Grid is considered as an abundant and common pool of resources with a varying state and the user has little knowledge about the current state of a Grid. The virtual pool of resources is dynamic and diverse in nature. New resources can be added to and some resources can be withdrawn from the Grid at any point of time. Thus the performance, as well as load changes frequently over time. Therefore, mapping the resource requirements onto the actual physical resources cannot be done at the user or at application level. In Grid environments, both users and resources appear differently at virtual and physical layers. Resources appear as entities distinct from the physical node in the virtual pool. Thus, in Grid, resources are selected first and then the mapping is done according to the resource selection. Different resource providers can satisfy resource requirement of a job in various ways. There must be an explicit assignment provided by the Grid environment between abstract resource requirements and physical resource objects. The actual mapping of jobs to nodes is driven by the resource assignment policies. A job is mapped onto a resource provider where its requirements can be satisfied. Once a resource provider is selected, a local process needs to be initiated on behalf of the user. However,

running a process is possible only for local users (account holders) on a particular node. Thus, Grid middleware performs an explicit mapping of an authenticated Grid user to a local user of the resource provider and authorizes the user to run processes on that resource provider [28, 31] (see next paragraph for details).

The most important distinction between a distributed system and a Grid environment is that the former comprises nodes that the user owns (meaning that the user has unrestricted access to the nodes), whereas the latter is based on the idea of resource sharing. By the term "sharing" it is meant that the user has no long-term access right (e.g. login access) to the resources and has the utilisation right only on a temporary basis. Because of the enormous size of the Grid and for its dynamic, resource-centric view, it is unrealistic to consider that a user has login account on every member node of the Grid simultaneously. Users may be authorized to temporarily have the rights of a local user for running processes on the node. Mere login-name and password-based authentication mechanism is therefore not sufficient for Grid environment. Every user on the Grid is identified via a certificate, which contains identification and authentication information for the Grid users [28]. Two different types of certificates are used in Grid, host certificate and user certificate. Certificate Authority (CA) issues certificate to hosts and users in Grid. Every user and host in Grid sends certificate-signing request to CA. CA signs that certificate using its private key and sends it back to its owner. These certificates are later used by the security system on the remote resource provider for authentication using the public keys.

For all the above differences between a conventional distributed environment and a Grid environment, the information services, the resource management and the security mechanisms that need to be maintained in a Grid environment are intricate and complex and add overheads to the overall performance of an application. A middleware is required to manage such an environment.

Next section highlights the importance of Grid middleware and describes the architecture and the infrastructure to be supported by any particular Grid middleware. In the last part of

this section, an existing Grid middleware has been considered as an example and its different components are described in order to obtain a clear understanding.

2.4 Overview of Grid Middleware

The mid-level software that provides services to users and to the applications in the Grid environment is called *middleware*. In the previous section, it has been pointed out that resource management in a Grid cannot be handled at the user or at application level. Therefore, a layer of software needs to be introduced between the application level and the system software level for managing heterogeneous resources, so that these resources can be accessed remotely by client applications without having any prior knowledge about the systems configurations. In addition, as the users work in collaborative environment under distinct administrative domains, services like security, data access, instrumentation, policy maintenance, accounting, and other special services required for applications, users, and resource providers to operate effectively in a Grid environment need to be provided at the middleware level. Thus, middleware acts as a sort of 'glue' which binds these services together.

Grid middleware are software stacks designed to present different compute and data resources in a uniform manner. The *middleware stack* is a series of cooperating programs, protocols and agents designed to help users accessing the resources of the Grid. An appropriate Grid middleware must provide a reliable implementation of the fundamental Grid services, such as information services, job management services which include job submission and monitoring, resource management services which include resource discovery, monitoring and brokering and data management services. The complexity of accessing the hardware and the software resources are entirely managed by the Grid middleware. Thus, the intention of the Grid middleware is to create virtual organizations with the resources, provide access while maintaining the policies at different levels, and in general deal with the physical characteristics of the Grid. In order to reduce complexity, Grid middleware should allow users and applications to access Grid resources in a transparent manner, the user does not need to know where the resource is physically

located, and the type of machine it is on. From a user perspective, therefore, Grids are all about resource accessing across sites and organizations without dealing with complexities related to management and security policies, accessibility options and other such type of issues.

The existing Grid middlewares are built as layered interacting packages and controlled by different managers called by a common API. The users are not concerned with the different syntaxes and access methods of specific packages. The user can simply submit its job through the API to the job manager.

The remaining part of this section focuses on an existing Grid middleware, namely the Globus Toolkit and describes how the important Grid services are implemented in the toolkit. However, before discussing the implementation of Globus, the Open Grid Services Architecture (OGSA) [33] is introduced in Section 2.4.1. The Open Grid Services Architecture (OGSA) describes an architecture for a service-oriented Grid computing environment for business and scientific use. OGSA has been described as a refinement of the emerging Web Services architecture, specifically designed to support the Grid requirements. OGSA has been adopted as Grid architecture by a number of Grid projects including the Globus Alliance. Therefore, the architecture is discussed briefly in the following section in order to obtain a clear understanding about the implementation of Globus.

2.4.1 OGSA and WSRF

The Open Grid Services Architecture (OGSA) [28, 33] developed by the Global Grid Forum (GGF) [40], aims at defining a common, standard, and open architecture for Grid-based applications. The objective of OGSA is to standardize practically all the services, which are commonly used for executing an application in Grid environment. These include job management services, resource management services, security services, etc. OGSA specifies a set of standard interfaces and depends on web services as the underlying middleware. OGSA first spawned the Open Grid Services Infrastructure (OGSI) [108],

which, despite the improvements of web services in several ways, failed to converge with existing web services standards. Although the web services architecture was certainly the best option, but it still failed to meet one of the most important requirements of OGSA - the underlying middleware had to be stateful. Web services, though in theory can be both stateless or stateful, they are usually stateless and there is no standard way of making them stateful. With an objective to fulfill this need, a new set of specifications for web services (published by OASIS [76]) have been introduced with major contributions from the Globus Alliance and IBM. These specifications are defined as Web Services Resource Framework (WSRF) [35] and form the infrastructure on which OGSA – the Grid architecture is built on. WSRF defines web service conventions to enable the discovery of stateful services [34] and interactions with them in standard and interoperable ways. Thus, WSRF provides the stateful services that OGSA needs. WSRF basically improved on OGSI and eventually replaced it. Figure 2.2 shows the relationship between OGSA, WSRF and Web Services.

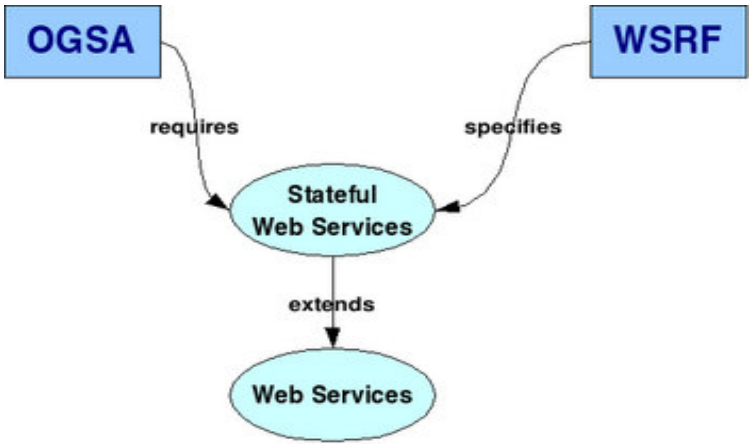


Figure 2.2: OGSA, WSRF and Web Services [98].

2.4.2 Globus Toolkit

The Globus Toolkit [41], developed by the Globus Alliance, is a Grid middleware constructed from a number of components that make up a toolkit. Globus includes quite a few high-level services that can be used to build Grid applications. These services, meet most of the abstract requirements set forth in OGSA. Most of these services are

implemented on top of WSRF (the toolkit also includes some services that are not implemented on top of WSRF and are called the non-WS components). The earlier versions including Globus Toolkit 3 were the OGSI-based release of Globus Toolkit. The version released at the end of April 2005, Globus Toolkit 4 (GT4) [47] includes a complete implementation of the WSRF specification as shown in Figure 2.3 [98].

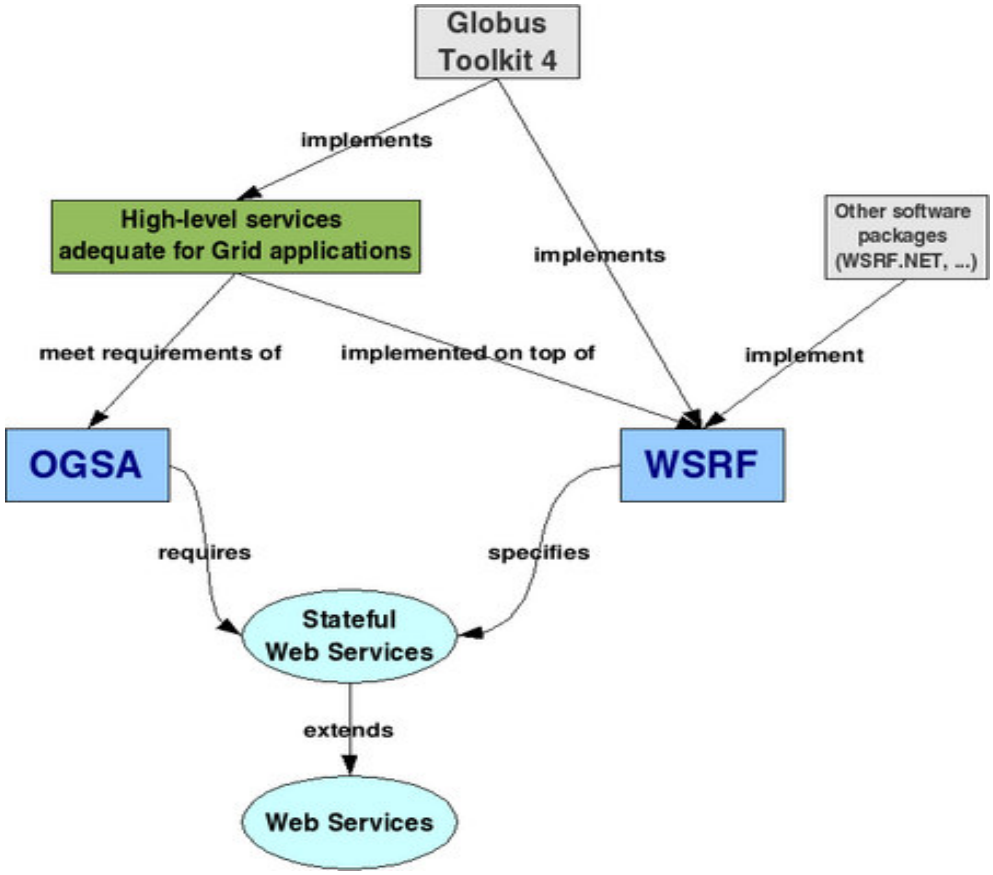


Figure 2.3: Globus Toolkit 4, OGSA, WSRF and Web Services [98].

The toolkit provides client, server and development components for the three Globus *"Pyramids"* [28] of Grid Computing: Resource management, Information Management and Data Management. Globus Toolkit GT4 [47] has these three pyramids of support built on top of a security infrastructure, namely Grid Security Infrastructure (GSI), as shown in Figure 2.4. These three pyramids provide support for resource management, data

management and information management and GSI provides different security services like authentication and authorization.

Globus is implemented with computers on a Grid that are networked and running applications. They also handle sensitive or extremely valuable data, therefore the security component of Grid plays an important role. In addition, Globus implements the resource management, data management and information services as separate components. The resource management component provides support for resource allocation, job submission, managing job status and monitoring progress of the job. Information services provide support for collecting static and dynamic information about resources and for querying this information. This is based on the Lightweight Directory Access Protocol (LDAP). The data management component provides support for transferring files among machines in the Grid and for managing these transfers. Brief descriptions of each of the above four components (security, resource management, information management and data management) are given below.

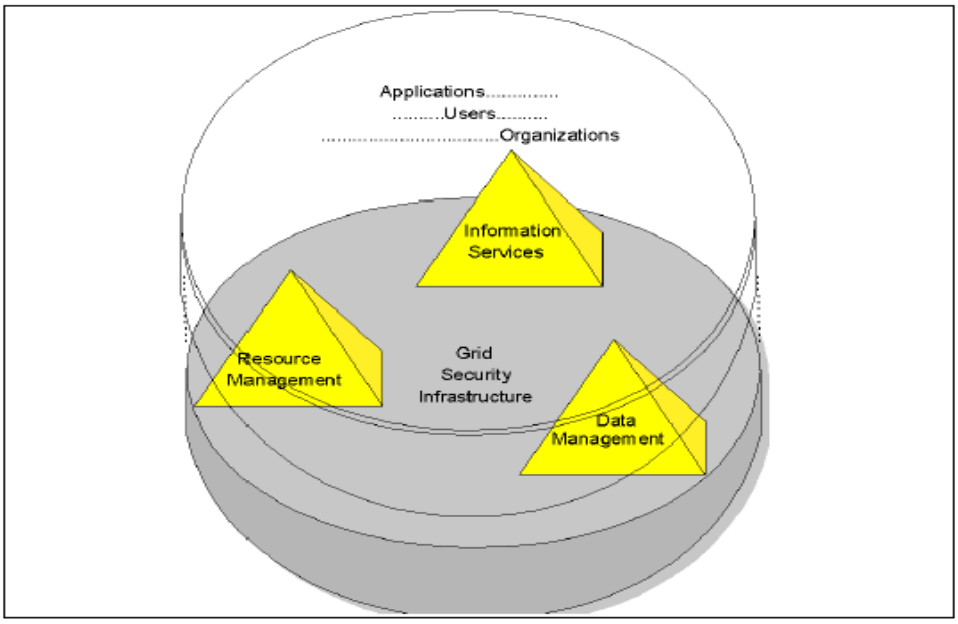


Figure 2.4: Three pyramids of Globus Toolkit GT4 [28].

Security

The Grid Security Infrastructure (GSI) is the primary security mechanism used by the Globus Toolkit. GSI uses Single Sign On (SSO) mechanism based on Public Key Infrastructure (PKI) and allows resource consumers to access resources securely. GSI complies with the Generic Security Service Application Programming Interface (GSS-API, RFC 2078/2743) standard [31]. The implementation of GSI uses X.509 [47] certificates, OpenSSL [79] libraries and the private keys as a credential for authentication. GSI includes both resource and client side tools for Grid security. On the resource side, GSI tools include X.509 credentials for identifying the resources. Every entity (user or client, host) accessing Grid resources must have its unique identity. The identity of the entity could be represented using certificates. These certificates are issued by a trusted Certification Authority (CA). The certificate (based on X.509) consists of some basic information including number and version of the certificate, name and ID of the entity and the issuer, public key of the entity, the validity period of the certificate, extensions of the certificate, the identifier of the signing algorithm, and the digital signature of the CA. On the client side, tools include facilities to create temporary credentials, called a *proxy.* Proxy is the entity that acts on behalf of a user (client) on a remote resource provider. The proxy certificate is based on the X.509 certificate. A proxy is generated with a new certificate valid only for limited time, signed using the user's certificate (user's private key). Proxy sends the user's certificate and the proxy certificate to the remote resource provider. The remote resource provider can verify the proxy certificate first using the user's signature (user's public key) and then using the signature of the CA (CA's public key), which is trusted by the remote resource provider. The remote resource provider gets the proxy's subject and the proxy's public key from the proxy's certificate using user's public key. The subject is a Distinguished Name (DN). The subject of the proxy certificate is similar to its owner's (user's) subject. In this way the remote resource provider also authenticates the user (client) standing behind the proxy. The user (client) performs mutual authentication for the target resource using the certificates and establishes a secure, encrypted communication channel after this initial handshake. The above procedure is similar to the one involved in Transport Layer Security protocol (TLS) that supports transport security HTTPS (based on

HTTP over SSL) and Secure Shell (SSH). While sending a request message for a job, the proxy encrypts the message using its private key and sends it to the remote resource provider. Remote resource provider decrypts the encrypted message using the proxy's public key and gets the request. Resource provider runs the request under the authority of a local user. Thus, the last step to the authorized usage of a resource after authentication is mapping the global identifier or name of a user to local user. The global name is based on the subject of the proxy certificate in the format of X.509 Distinguished Name (DN). The Grid resources have public gridmaps, which contains the mappings between DNs and local user names [47].

Resource Management

The Grid environment comprises a large number of users and resources in different administrative domains. It has already been discussed that management of resources in the Grid environment becomes complex, as the resources are geographically distributed, heterogeneous in nature, owned by different individuals/organizations. From the users' point of view the whole Grid should be seen as a single computer with appropriate software, hiding all the technical details related with physical locations, middleware, operating system, etc. The major difficulty in providing such a view of the Grid environment is its natural dynamicity; any node can join or leave the Grid at any moment of time. Thus, mapping the resource requirements onto the actual physical resources is not easy to manage at the user level. Therefore, mapping of the resources from the virtual level to the physical level is explicitly handled by the middleware. The Grid Resource Allocation Manager (GRAM) works with this objective and is responsible for Grid resource allocation and management and allows one point access to remote resources [28]. Gatekeeper, Job Manager and GASS are the main components of GRAM. The gatekeeper daemon builds the secure communication between clients and servers. The gatekeeper is responsible for authenticating and authorizing a Grid user. The authorization is based on the users' Grid credential and an access control list contained in a configuration file known as the gridmap file. When a client submits a job, the request is sent to the remote host and handled by the Gatekeeper daemon located in the remote host. The specification of job requirements is

written by the client in a language called Resource Specification Language (RSL) and is processed by GRAM as part of the job request. GRAM processes the requests for resources for remote application execution, allocates the required resources, and manages the active jobs. The Gatekeeper creates a Job Manager to start and monitor an active job. When the job is finished, the Job Manager sends the status information back to the client and terminates. In earlier versions of Globus, GASS (Global Access to Secondary Storage) is used for moving executables between storage servers and the execution nodes, moving input data to the execution nodes and providing mechanisms for transferring the output files to clients. The latest version of Globus Toolkit, GT4 [47] implements WS-GRAM, which is based on WSRF and provides a set of services to allow accesses to computing resources through web services conforming to WSRF model. GASS is removed from this implementation and only GridFTP and Reliable File Transfer (RFT) are used, thereby reducing the overhead.

Information Management

Management of Grid resource information is essential, but complex due to the dynamic nature of Grid and wide range of resources that should be represented. Monitoring and Discovery Service (MDS) of Globus provides access to static and dynamic information regarding resources [28, 47]. It comprises two services, namely Grid Resource Information Service (GRIS) and Grid Index Information Service (GIIS) and multiple MDS clients. GRIS registers its local information with the GIIS. In addition, one or more information providers can be used in order to obtain resource information in a comprehensive manner. The information providers obtain the resource information and pass that to GRIS. MDS clients can get the resource information directly from GRIS for local resource information. The Grid Resource Information Service (GRIS) and Grid Index Information Service (GIIS) can be configured in a hierarchy to collect the resource information and distribute it. Thus a GIIS at a particular site may register with another GIIS and become part of the hierarchy. MDS clients access these GIIS for Grid wide resource information. Both, GRIS and GIIS are based on the Lightweight Directory Access Protocol (LDAP). MDS2 (pre-web services MDS), developed with GT2, is an LDAP-based implementation of information services.

This component is also available in later versions of the Globus Toolkit (GT3.0, GT3.2 and GT4). Additionally, the Globus Toolkit version 4 (GT4) contains MDS4 or WS MDS, which implements WSRF [47]. MDS4 supports Ganglia and Hawkeye as information providers. MDS4 consists of three components that are used to gather, manage, index, and respond to queries regarding resources and computational status. These are WS MDS Index service, WS MDS Trigger service and WS MDS Aggregator. WS MDS Index service collects status information from Grid resources and makes it accessible in one location. Generally, a VO deploys one or more index services for collecting data about all Grid resources available within that VO. WS MDS Trigger service collects data from Grid resources and passes the data to appropriate programs to perform various actions in response to events, as specified by the administrator. WS MDS Aggregator is the software framework on which the other two services are built.

Data Management

In the Grid community, there is a popular expression that "access to the data is as important as access to the compute resources". There are two components related to data transfer in the Globus Toolkit: the Globus GridFTP tool and the Globus Reliable File Transfer (RFT) service [47]. GridFTP is used for the secure, robust, fast, efficient and high performance data transfer. GridFTP protocol navigates different types of storage systems, has a number of compelling new parallel and striped data transfer capabilities, and includes various new instrumentation and TCP buffer features. The GridFTP component uses a non-WS protocol, which started as an ad hoc Globus protocol, but later became a GGF [40] specification. In general, GridFTP is a very powerful tool, but it requires an open socket connection between the client and server throughout the transfer. This may not be convenient for long transfer. RFT is a WSRF compliant web service that offers the capability of requesting file transfers in disconnected mode. The client first requests the transfer and then hands over the responsibility for managing it to the RFT service. The RFT service uses a persistent database to record transfer requests as well as the progress status of transfers. If the RFT service fails for some reason, it resumes the incomplete transfer when it is restarted. It

restarts each transfer at the point where the status was last recorded rather than restarting at the beginning of the transfer.

Figure 2.5 shows the architecture of GT4 and its components. The components are divided into five categories: Security, Data Management, Execution Management, Information Services and Common Runtime.

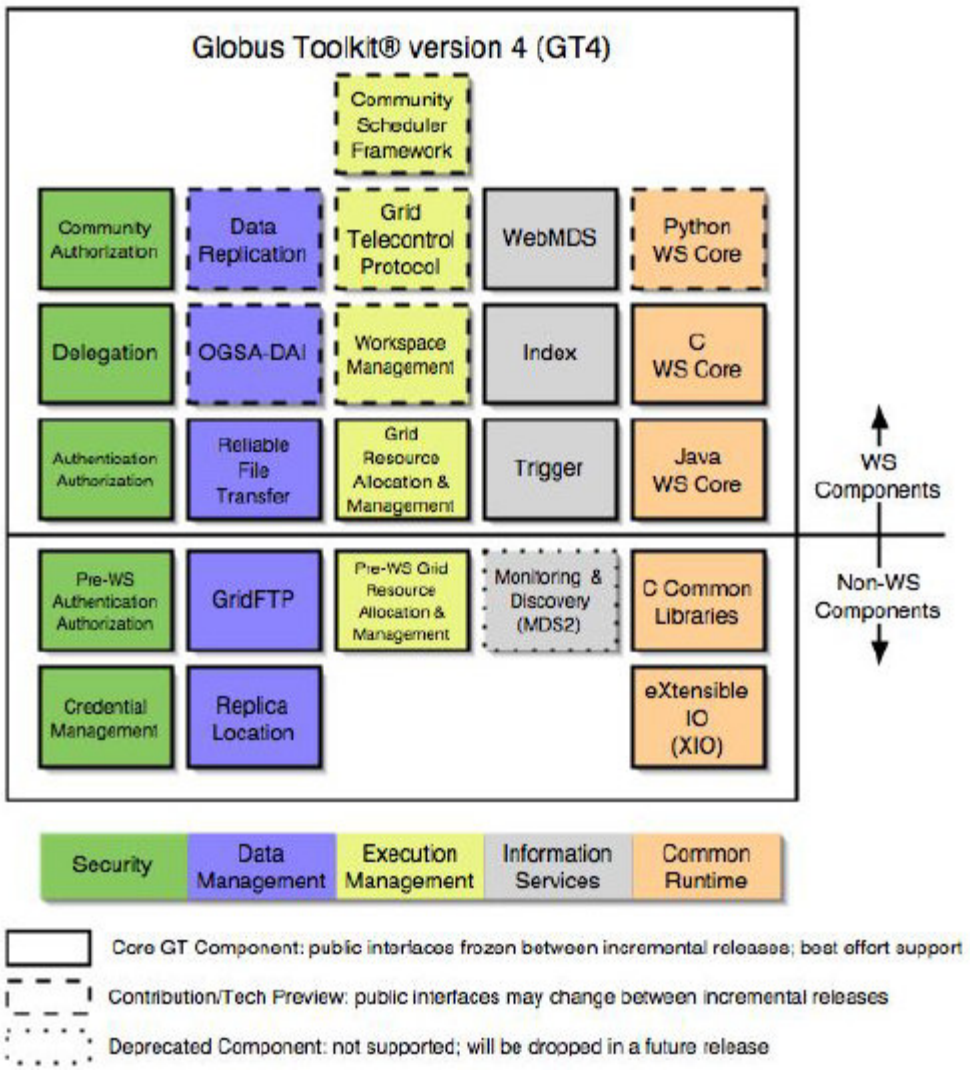


Figure 2.5: GT4 architecture [98].

2.5 Service Level Agreements and Quality of Services

An important aspect of resource management in Grid is resource brokering. The resource broker finds out which of all the available resources fits the best to the requirements of a job. For example, if the job requires some specific platform or a special resource, the resource broker takes care of selecting the best resource provider with that kind of resource and allocates the job onto the resource provider. Service Level Agreements (SLAs) provide a mechanism to correspond with the capability and the status of resources offered by the resource provider. When a suitable resource provider is selected for a specific job, the resource broker establishes an SLA between the client and that resource provider. The ability to control and manage the Quality of Service (QoS) is a crucial challenge for resource providers and is also the responsibility of a Grid resource management system. Thus how much of the job requirements can be fulfilled by the selected resource provider must be clearly specified in the SLA. Generally both resource providers and clients desire a customized SLA.

Some definitions of SLA as stated in [49] are given below:
"An agreement between two entities determining the extent and degree of assistance that will be provided."
"An agreement between two parties defining mutual expectations."
"A contract between a provider of service and a consumer of service which establishes measurable agreed-to targets of performance."

SLAs are negotiated between the client, the resource broker and resource providers, to decide all types of expectations and obligations which include job start and end times, expected completion time of the job, amount and status of resources and metrics needed to check the QoS, and may be re-negotiated during runtime. Resource broker obtains the job requirements from a client and contacts resource providers, which may support these requirements. The resource broker gathers information on the current state (e.g., current load) of the resource providers and on the basis of this information, a decision is made regarding selection of appropriate resource provider(s) that can be effectively used to run

the job. This decision is followed by a negotiation with these resource providers. As discussed in [24], the negotiation is based on a framework provided by the Service Negotiation and Acquisition Protocol (SNAP) and thereby guarantees are obtained that the user's requirements will be fulfilled by the use of a Service Level Agreement (SLA). It is expected that SNAP protocol will also be implemented within OGSA [23, 24]. SNAP comprises three main SLAs, namely Task Service Level Agreement (TSLA), Resource Service Level Agreement (RSLA) and Binding Service Level Agreement (BSLA). TSLA is defined when the users provide clear objective specifications of their task requirements and any resource preferences. A resource provider promises its offers in the RSLA, which relates to resource discovery and decision-making regarding the appropriate resources that meet the requirements of a task and assists in securing the resources for utilisation. BSLA only associates the task with the resources.

An SLA specifies the agreed upon level of availability, serviceability, performance, and operation both in business value terms understood by end-users, and technical terms that can be enforced to reserve resource capabilities. Typically, an SLA contains Quality of Service commitments including penalties and rewards, pricing policies, authorization policies, and negotiation policies. To ensure the authenticity of an SLA it is digitally signed by all parties using a trusted third party (TTP)-based model. Client, resource provider and resource broker collectively construct an SLA. Client embodies a request for resources and the resource provider represents an offer for available resources and guarantees their quality of services in terms of availability and performance. When the resource requirement of the client matches with the offer given by a resource provider, the resource broker prepares an agreement between the client and the resource provider. Both the client and the resource provider need to start by gathering information so that each has a concrete basis on which to negotiate. Before eliciting commitments from the resource providers, clients should carefully review and clarify their resource needs. Also before making any commitments to clients, resource providers should determine the levels of services they can realistically provide. Thus, this thesis proposes to implement this as a two-way agreement, one agreement between the client and the resource broker and the other one between the

resource broker and the resource provider. Once the SLA is mutually agreed upon and signed, it should be actively managed to ensure that all the commitments are attained.

Shi Z. et al proposes [97] that SLA can be used as a means of managing and monitoring Quality of Service (QoS) at two levels, namely guaranteed-service and controlled load, and to enforce contracts. If the client requires "Guaranteed-service" QoS, then the SLA consists of exact constraint that must be met by the resource provider. If a "Controlled-load" QoS is requested, then the QoS requirements are more relaxed and the parameters are specified as a range constraint.

A typical structure and elements of a SLA specification can be found in [59, 83]. It gives indication of the components, which make up the SLA. A similar structure with minor modification will be used in this thesis as shown in Table 2.1.

Component	**Observation**
Purpose	Run a Grid job with guarantees to ensure user's requirements are met.
Parties	The client, the resource provider and the resource broker.
Scope	Computational Service, Information Service etc.
Service Level Objective	Ensure availability of resources that satisfy jobs requirements for the duration of the job execution.
SLO Attributes	Number of Processor, CPU speed, available free memory, operating system etc.
SLIs	For each SLO attribute, its value is a service level indicator.
Administration	The SLA's objectives are met through resource brokering.

Table 2.1: SLA specification.

The SLOs (Service Level Objectives) represent a qualitative guarantee such as number_of_Processor, CPU_speed or avail_free_memory etc. They comprise a set of SLI

(Service Level Indicator) parameters, which represent quantitative data describing the level of the SLO guarantee as shown in Figure 2.6.

```
<SLA>
      <Party>
            <Client_ID>XXX</Client_ID>
            <ResourceProvider>YYY</ResourceProvider>
            <ResourceBroker>ZZZ</ResourceBroker>
      </Party>
             <job_type> computational </job_type>
             <start_time>10-02-2007:8:00:00</start_time>
             <end_time>10-02-2007:10:00:00</end_time>
     <resource_avail>
            <metric>
                  <SLO>num_processor</SLO>
                  <SLI>4</SLI>
             </metric>
             <metric>
                  <SLO>os</SLO>
                  <SLI>linux</SLI>
             </metric>
     </resource_avail >
      <QoS_class> Guaranteed-Service </QoS_class>
</SLA>
```

Figure 2.6: Example of SLA.

2.6 Conclusion

This chapter presented an overview of the Grid environment and discussed different services provided by the Grid middleware. Existing Grid middleware provides functionalities to manage resources in the dynamic, distributed, heterogeneous and complex Grid environment. An existing middleware, namely Globus Toolkit has been briefly described in order to develop an understanding about the important Grid services including resource management service, information service, data management service and security service. In Globus, the first three services are built on top of the security infrastructure GSI.

Although Globus offers the basic services in a Grid environment, no support is provided to the client for selection of the right resource provider on the basis of the requirements of a

job or to make a cost effective selection of the resource provider if more than one resource providers are available to fulfill the need of a particular job. Also no strategy has been developed to handle the execution of multiple concurrent jobs. Resource brokering on the basis of service level agreement (SLA) and negotiation facilities to improve the quality of services is also not supported by Globus. Once a job is submitted, it is important to monitor the performance of the job and also the infrastructure till the completion of the job so that the required performance can be achieved. Generally, Globus monitors the job progress periodically but takes no action in case of performance degradation.

This thesis puts emphasis on the requirement of a performance based resource management system, which can be built on top of the existing middleware. The main focus of this thesis is achieving high performance while executing an application in Grid.

In order to understand the requirements of a resource management system in high performance computational Grid environment, the next chapter reviews some of the existing Grid resource management systems, which extend the functionality of GRAM in Globus. On the basis of this study, an integrated framework for resource management is introduced in Chapter 3, which is elaborated in the later chapters.

Chapter 3

Resource Management

The previous chapter has drawn attention to the requirement of an efficient resource management system in Grid that provides an execution environment for high performance complex applications. Resource management systems handle pool of resources that are available to the Grid. Resource management systems for Grid must support adaptability towards its heterogeneous and dynamic nature, extensibility to multiple administrative domains, job submission, resource discovery and selection, and maintain quality of services and resource cost constraints. Resource management systems need to oversee the performance of the job after submission in addition to performance monitoring of the infrastructure. Resource management systems must also be able to improve the performance of the job in case of any performance problem. The thesis argues that a Grid resource management system should take care of all the above-mentioned features.

One of the main functionality of the resource management system is to assign resources to jobs in response to resource requests that a client makes during the job submission – the process is called resource brokering. Many of the existing resource management systems concentrate on resource brokering. This chapter first focuses on the other researchers' contribution in resource management systems and discusses the strengths and limitations of various such systems. In the later part of this chapter, a framework is introduced which handles related issues and attempts to fulfill the requirements as recognized in this thesis.

3.1 Existing Resource Management Systems – A Review

This section briefly describes some of the existing resource management systems, which mainly act as resource brokers in the environment. As mentioned earlier, the purpose of a broker is to dynamically find, characterize and allocate the resources most suitable to the user's applications. Some of the systems also incorporate additional functionalities of resource management. This section does not aim at providing an exhaustive survey of the existing resource management systems. One such detailed survey has been carried out by Krauter et al [61]. This section only focuses on some representative systems and discusses the approaches used by them along with their shortcomings.

3.1.1 Condor-G

Condor-G works with Grid resources, allowing users to submit jobs, manage jobs, and get jobs executed on widely distributed machines. Condor-G [21] combines the strengths of both Condor [20] and Globus [41]. Condor handles job submission within a single administrative domain, whereas Globus manages job submission within multiple administrative domains. Condor-G system handles security, resource discovery, and resource access in multi-domain environments, as supported within the Globus Toolkit. In addition, management of computation and harnessing of resources within a single administrative domain, as performed within Condor, are also supported by Condor-G [55]. The inter-domain resource management protocols of the Globus Toolkit and the intra-domain resource management methods of Condor are combined to allow the user to harness multi-domain resources as if they all belong to one personal domain. The user defines the tasks to be executed. Condor-G handles all aspects of discovering and acquiring appropriate resources, regardless of their location. It claims to be a complete Work Load Management System (WMS) for Grid resources. The system performs the matchmaking of jobs to resources based on requirements and preferences for static and dynamic resource attributes. Condor-G is an extension of the Condor Batch System. In the Condor Batch System (BS), each *Worker Node* (WN) (a processing unit) publishes its capabilities and status to a *Collector* (a daemon responsible for collecting information about the status of the available resources) in the form of a ClassAd, which is a list of attribute value pairs. A user job is

also represented by a ClassAd and is also sent to the *Collector*. It contains the requirements and preferences of the job in terms of WN capabilities e.g., OS or minimum RAM. A *Negotiator* periodically processes the job and resource ClassAds to match jobs to the most appropriate WN. Another component, *Scheduler* handles job submission and transfers the job's executable and standard input files using Globus services. Condor-G provides a client implementation of the GRAM protocol (see Section 2.4.2). Figure 3.1 shows the architecture of Condor-G including the three condor services.

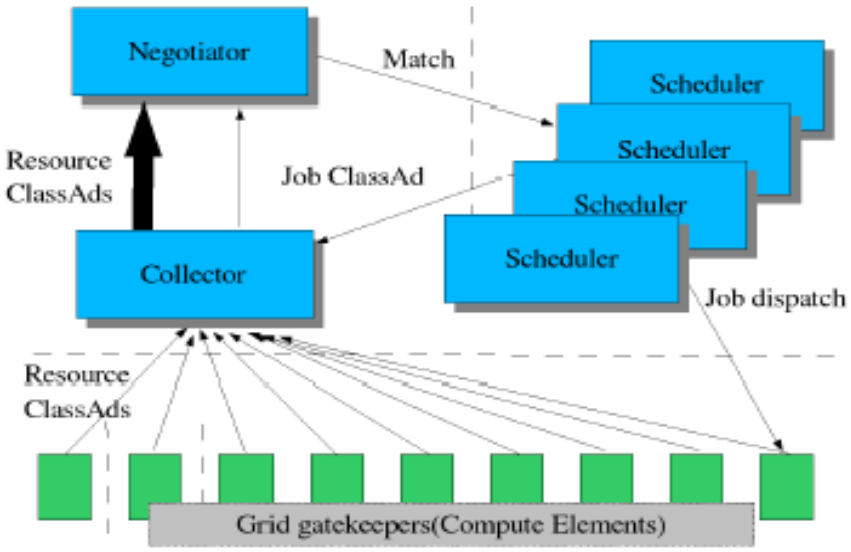


Figure 3.1: The service architecture of Condor-G [111].

On behalf of client, Condor-G computation management service (or Condor-G agent) executes user computations on remote resources. It does this by using the Grid protocols like GRAM, GASS, and GSI. In particular, it transfers a job's standard I/O and executable using GASS, submits a job to a remote node using the revised GRAM job request protocol, and subsequently monitors job status using the GRAM protocol, while authenticating all requests via GSI mechanisms. The Condor-G agent also handles resubmission of failed jobs, communications with the user concerning unusual and erroneous conditions and recording of the computation on stable storage to support restart in the event of its failure.

Figure 3.2 depicts how Condor-G supports remote execution. The *Scheduler* receives request from user to submit its jobs to its intended Grid resources. The *Scheduler* creates a new GridManager daemon for submitting and managing these jobs. One GridManager handles all jobs for a single user and it terminates when all submitted jobs complete. Each GridManager job submission request results in the creation of one Globus JobManager daemon. This daemon connects to the GridManager using GASS in order to transfer the executable and standard input files, and subsequently to provide real-time streaming of standard output and errors. The JobManager then submits the jobs to the local scheduling system of the execution site. The JobManager sends back the current job status back to the GridManager, which then informs the *Scheduler*, where the job status is stored persistently.

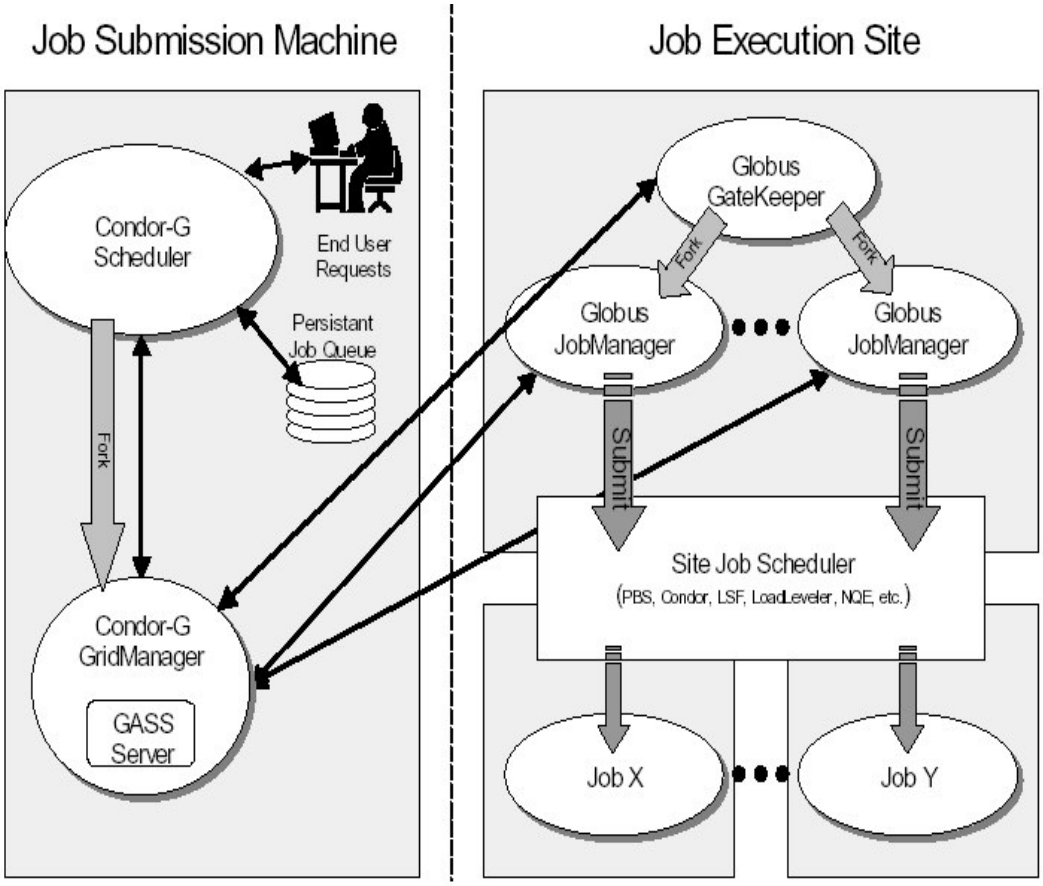


Figure 3.2: Remote execution by Condor-G [55].

Condor-G has the following limitations:

- Condor-G requires the user to specify explicitly the resource where to run a job at the time of job submission. That is, Condor-G does not assist the users to select appropriate Grid resources and does not really allow them to take advantage of Grid environment, which consists of many different resources with different capabilities.
- Once a job is allocated onto a remote node then rescheduling elsewhere is difficult in Condor-G.
- Condor-G does not use any cost optimization technique during the selection of appropriate resource provider for a job or a batch of jobs.
- Condor-G monitors the execution of individual jobs and takes necessary actions for resubmission of failed jobs. However, no particular approach has been adopted for monitoring the concurrent execution of multiple jobs. Also, many issues related to QoS maintenance have not been considered.

3.1.2 Nimrod-G

Nimrod-G [75] has been developed as a Grid resource broker based on the GRACE (Grid Architecture for Computational Economy) framework [43]. It is a tool for automated modeling and execution of parameter sweep applications (parameter studies) over global computational Grids. It leverages services provided by Grid Middleware systems such as Globus [41], Legion [17, 63], and the GRACE trading mechanisms [11, 43]. A set of protocols for secure and uniform access to remote resources is provided by the middleware system. The middleware system also provides services for accessing resource information and storage management.

The Nimrod-G [12] system automates the allocation of resources and application scheduling on the Grid using economic principles, and scheduling optimizations. Figure 3.3 shows the modular and layered architecture of Nimrod-G. The two main components of Nimrod–G are Nimrod-G Clients and the Nimrod-G Resource Broker. Parametric modeling is concerned with computing a function value with different input parameter values and makes it possible to explore different parameters and design options.

Parametric experiment generally means an executable program with number of design scenarios. In order to support the modeling and experiments, Nimrod-G client contains:

- **Tool for creating parameter sweep applications:** The tool consists of GUI based scripting mechanism and a declarative parametric modeling language for expressing parametric experiments.
- **Steering and Control Monitor:** A user-interface for controlling and supervising the experiment under consideration is used for varying the parameters related to time and cost and to influence the scheduler to take decision while selecting resources. A user can also view the status of all jobs using this interface.

In addition to regular applications which are modeled using scripting tools, specialized applications can also be executed which use Nimrod-G job management APIs for adding and managing jobs.

Nimrod-G Resource Broker [12] contains:

- **Task Farming Engine:** This is the central component from which the entire experiment is managed and maintained. Task Farming Engine can be used for creating and plugging user defined scheduling policies and customized task-farming applications. The task farming engine coordinates resource trading, scheduling, staging data, execution and accumulation of results from remote Grid nodes. The farming engine is responsible for parameterization of the experiment, actual creation of jobs and maintenance of job status. Nimrod-G clients add jobs to farming engine that works with scheduler and dispatcher to complete the experiment. The farming engine also maintains the state of the whole experiment and ensures that the state is recorded in persistent storage.
- **Scheduler:** The scheduler performs resource discovery, trading and scheduling. It interacts with a Grid information service MDS (in Globus) to get the list of authorized machines. Its other responsibilities include negotiation for resource access costs and keeping track of resource status information. It selects resources that meet the deadline and minimize the cost of computation.

- **Dispatcher and Actuators**: It starts the execution of a job on the selected resource as instructed by the scheduler. Farming engine, after obtaining the instructions from the scheduler, informs the dispatcher to map an application task to the selected resource. The dispatcher triggers appropriate actuators depending on the type of the middleware running on resources to deploy resource-mapped jobs on Grid-resources. For example, a Globus specific actuator is required for Globus resources, and a Legion specific actuator is required for Legion resources.
- **Nimrod-G Agent:** Nimrod-G resource broker deploys agents for managing the execution of jobs on resources.

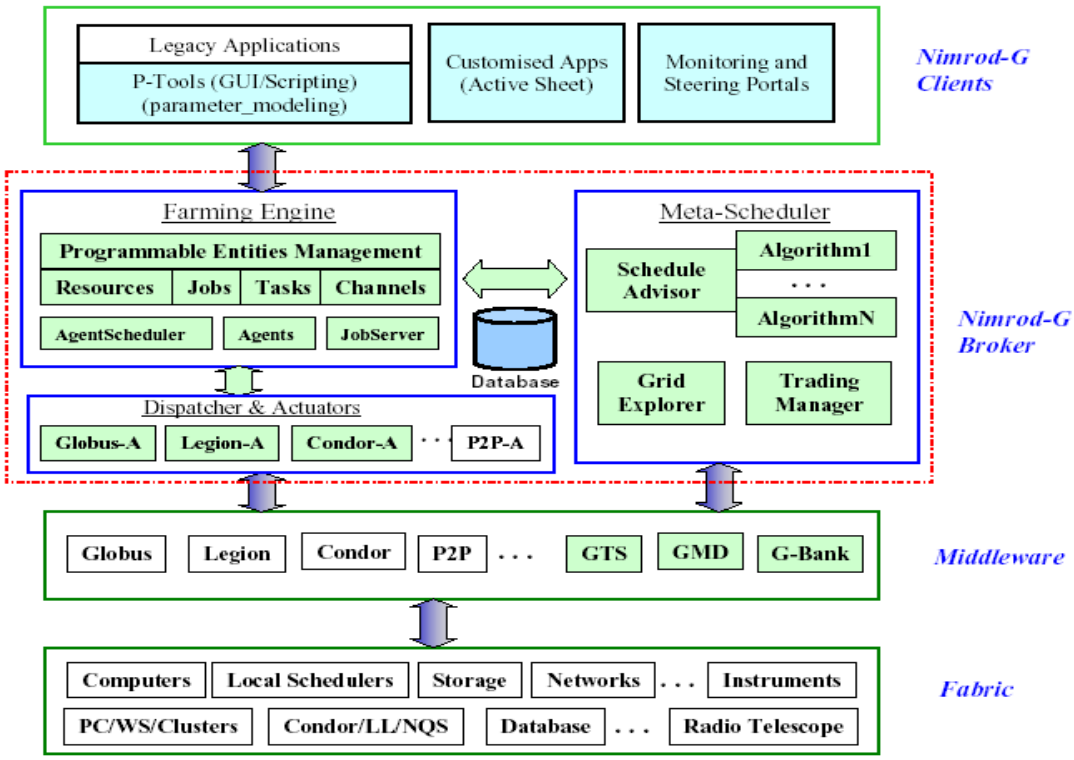


Figure 3.3: Layered architecture of Nimrod-G [12].

Nimrod-G incorporates four adaptive algorithms for deadline and budget constrained scheduling. These algorithms are based on cost optimization, time optimization, cost-time optimization and conservative time optimization strategies respectively. The objective of

the cost optimization technique is to minimize the execution cost. On the other hand, the time optimization technique aims at minimizing the time. The cost-time optimization is similar to cost optimization, but if there are multiple resources with the same cost, it applies time optimization strategy while scheduling jobs on them. The conservative time optimization technique takes care of the deadline and budget.

Nimrod-G has the following drawbacks:

- Nimrod-G provides an execution facility for the parametric problems, which are only a small share of the Grid applications.
- Cost optimization technique is used to minimize the execution cost of a particular job; optimizing the overall cost for executing a batch of jobs has not been considered.

Nimrod-G maintains the deadline and budget constraints for a particular job as defined by the user. Nimrod-G monitors the probability of successful completion of a job, but does not monitor the job performance and the performance of the execution environment. Like Condor-G, if the performance of a job is not satisfactory, Nimrod-G does nothing to improve its performance.

3.1.3 SNAP Resource Broker in White Rose Grid

The White Rose Grid (WRG) [112] provides an infrastructure for experimenting with virtual resources of many types e.g., compute, storage, software, networking etc. as Grid services. An infrastructure such as WRG works with heterogeneous resources and spans over multiple administrative domains. The community resource broker on the WRG enables transparent submission of jobs to the Grid. The users submit their requirements (e.g., number of processors, operating system and job description) to the broker. The broker contacts the resources that may support these requirements and gathers information on their current state, e.g. current load. A decision is made regarding the resources that will be used to run the job and this is followed by a negotiation with these resources. This negotiation is based on the framework provided by the Service Negotiation and Acquisition Protocol (SNAP) [42, 48], which uses Service Level Agreements (SLAs) to guarantee that the

requirements of the users will be fulfilled. SNAP comprises three main SLAs, namely TSLA (Task Service Level Agreement), RSLA (Resource Service Level Agreement), and BSLA (Binding Service Level Agreement) (as discussed in Section 2.5). Figure 3.4 shows the components of a SNAP based resource broker. The topmost layer of SNAP is for parsing the user requirements submitted through a Grid portal. The second layer, on the basis of the parsed user requirements, uses a Matchmaker to contact a Knowledge Bank (KB). Knowledge Bank is a repository that stores static information regarding all resources. The broker can access this information on behalf of the user. The information stored in the KB includes the number of processors, the operating system, memory, storage capacity and past behaviour performance of a resource. On receiving the information, the Matchmaker forwards the details to the Decision Maker for finding potential resources for the application. Based on the information received from the Decision Maker, the Resource Gatherer queries the information provider on each selected resource and gathers the dynamic information about their status. When all queries are reported back to the Resource Gatherer, the information is forwarded to the Coordinator, which selects the resources for executing the applications and acquires them for utilisation through the use of immediate reservation. The final procedure is the part of the Binding Service Level Agreement (BSLA) and is executed by the Dispatcher by submitting the task and binding it to the resources.

A simple approach has been used in SNAP resource broker, although it lacks some of the important requirements for high performance applications as described below.

- Within this framework, Coordinator selects a resource provider for an application on the basis of the information available from the information provider. However, the strategy for selecting a resource provider when more than one resource providers are available to fulfill the requirements of a particular application has not been defined.
- The concept of SLA is used for resource brokering purposes, but the issues related to the maintenance of the QoS have not been dealt with.

- The resource broker does not use any cost optimization technique at the time selecting an appropriate resource provider for a job.
- The issue related to the concurrent execution of multiple jobs has not been addressed.

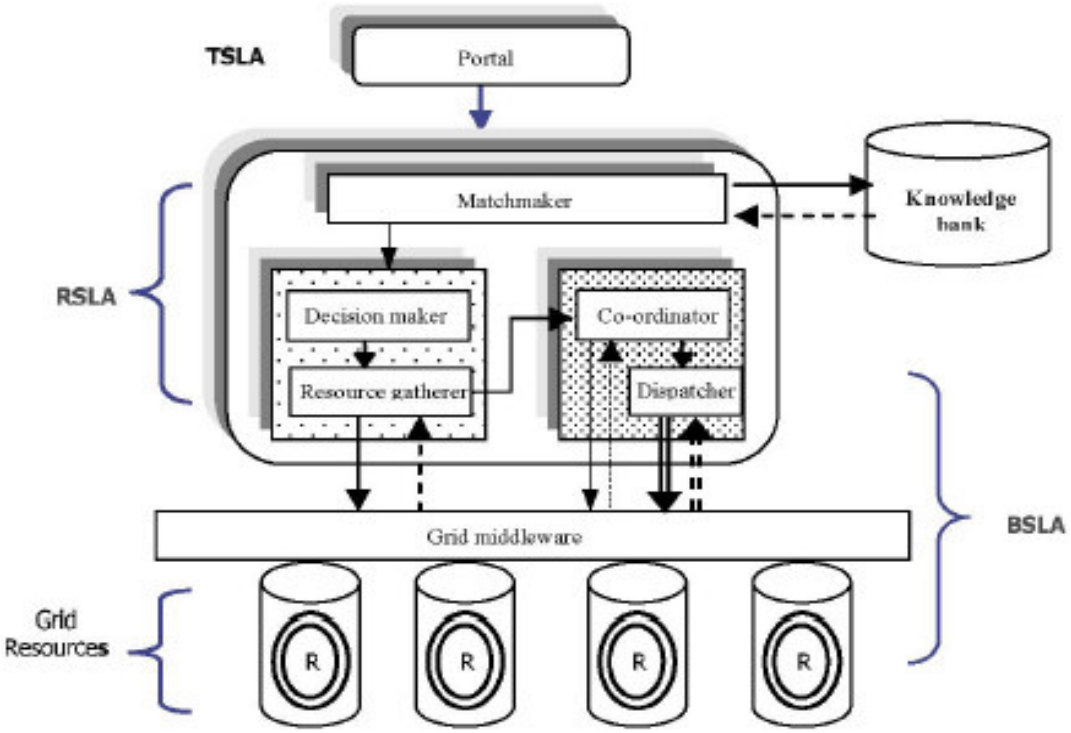


Figure 3.4: Grid resource broker architecture within the SNAP framework [42].

3.2 Performance-based Resource Management

The previous section has discussed the architecture and services offered by three existing resource management systems for Grid environment, these include Condor-G [21], Nimrod-G [75] and SNAP based resource broker in White Rose Grid [112]. All three resource management systems primarily focus on the resource brokering activity. However, they do not address issues related to achieving high performance of complex applications, nor do they attempt to enhance the performance of the jobs at run-time.

Condor-G manages job execution within multiple administrative domains using the capabilities of Condor and Globus. Nimrod-G also manages job execution within multiple administrative domains. SNAP resource broker within WRG efficiently handles job execution within multiple administrative domains as WRG infrastructure spans over multiple administrative domains. Condor-G and Nimrod-G both support multiple job submission. Condor-G can also handle inter-job dependencies and has fault tolerance capabilities, e.g. it can handle the situations when the network goes down or machine crashes. Nimrod-G maintains the deadline and budget constraints for a particular job and incorporates four adaptive algorithms for this purpose and optimizes time and cost. It also monitors the probability of successful completion of jobs. The concept of SLA is used in Nimrod-G and SNAP based resource broker.

In spite of all the significant contributions, the above mentioned resource management systems have some limitations as revealed in the earlier section. Some of the problems that have not been dealt with in these systems are summarized here. The resource brokers within these systems do not consider optimization of the total execution time (or cost) of a batch of jobs or the tasks of an application, although many of them handles multiple job submission. Nimrod-G ensures that an individual job is executed in minimum time (or with minimum cost) by selecting an appropriate resource provider based on some time and / or cost optimization algorithm. However, in case of a batch of jobs, the algorithm for minimizing the time / cost of individual jobs may not give the best effect (as will be discussed in Chapter 5). Instead, algorithms aiming at overall time / cost optimization for a batch of jobs should be developed for high performance of the applications. Performance monitoring of the jobs and the execution environment in order to maintain the QoS is another important requirement and this should be integrated with the resource management system. Moreover, jobs executing in a dynamic environment like Grid should adapt to any kind of changes in the environment. None of the above resource management systems has the flexibility to allow adaptive execution of applications running in Grid in order to achieve the desired level of performance. Existing resource management systems pay main attention to the resource brokering strategies at the time of initial job submission. This

thesis argues that solely focusing on initial resource allocation services an efficient Grid resource management system cannot be developed.

From the above discussion, it can be inferred that resource management in Grid environment targeting high performance of an application must involve executing the application or its components on the right resource providers all the time. A Grid resource broker should function with this objective. This objective can be achieved by the idea of collaborative usage of distributed resources and allocating and reallocating jobs on these resources even at runtime. This task is dependent on the dynamic information that is available after monitoring the current status of all the available resource providers. Moreover, only selection of the resource provider based on the requirements / performances of each individual job is not adequate for a Grid resource broker. This research work targets concurrent execution of multiple jobs (components) of an application on a set of distributed resources in order to benefit the application in terms of execution time. No effective strategy has so far been found in the existing resource management systems that targets optimization of the overall time / cost for a batch of jobs.

One more important aspect is that when a resource provider promises to deliver a QoS, the resource broker must ensure that it delivers the promised QoS. Thus, regular performance monitoring is another requirement of the resource management system. Some research works have pursued maintenance of performance QoS by monitoring the job performance. The ICENI project emphasizes a component framework [37] and makes use of run-time information regarding available resources and their capabilities along with the high-level information concerning the components. This information is passed to a scheduler, which selects the target resources and an optimal implementation of the concerned components. On the other hand, the GrADS project focuses on building a framework for both preparing and executing applications in Grid environment [22, 44]. Each application has an application manager, which monitors the performance of that application for QoS achievement. Failure to achieve QoS contract causes a rescheduling or redistribution of resources. The resources are monitored using NWS and Autopilot is used for performance

prediction [92, 115]. Although, the objectives of this thesis are somewhat similar to the goals of some of the research projects discussed here, the focus of this research work is only on the resource management and approaches taken up are very much different from the others. This research work puts emphasis on *resource brokering with the aim of overall time / cost optimization for batch of jobs* and *maintenance of performance QoS* in a resource management system for Grid. To maintain the performance QoS, resource management system can take run-time actions, which may include *rescheduling or local tuning* decisions.

On the basis of the requirements set forth in the previous paragraphs, this thesis proposes the design of an integrated framework for resource brokering, performance monitoring of jobs and adaptive execution capability for a single computational job or more than one concurrent jobs running in Grid environment. The major goals of the framework can be defined as

- Resource brokering with the aims of overall time / cost optimization,
- Regular monitoring of performance of the job and the infrastructure,
- Adaptive execution, which includes rescheduling and local performance tuning of jobs.

The framework assists in decision making for resource brokering activities on the basis of monitoring information gathered regarding the (run-time) performances of all the available resource providers. The framework provides support for maintaining the QoS offered by the resource providers. SLAs are established by the resource broker involving the client and the resource provider and QoS is maintained on the basis of these SLAs. Run-time actions may be taken depending on the performance monitoring data gathered on the resource providers. Thus, the entire working principle of a system built on top of this framework will be dependent on the performance monitoring of the jobs and the infrastructure. To summarize, the framework provides *performance based resource management ability for executing complex applications in the dynamic and heterogeneous environment like Grid.*

The performance based resource management framework proposed in this thesis comprises four different components, which work in an integrated manner. These four components are: (a) *a Resource Broker (b) a Job Controller (c) an Analyzer and (d) a Performance Tuner.* Figure 3.5 shows the component diagram of the framework.

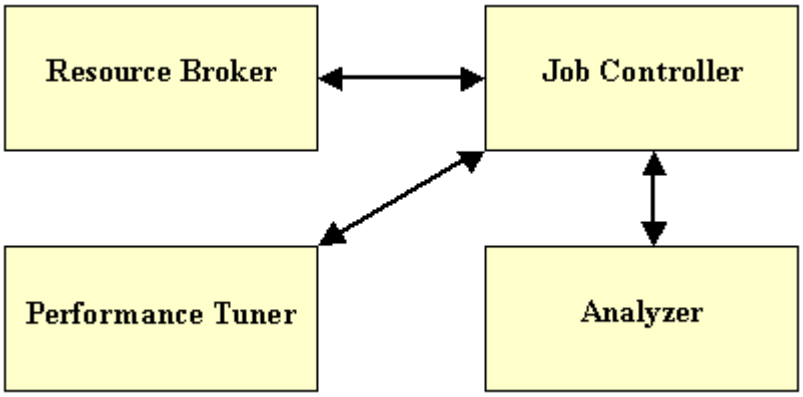


Figure 3.5: Components of the framework.

The thesis considers that an application is decomposed into multiple work items or tasks (referred to as jobs in the thesis). These jobs are executed concurrently on the Grid resources in order to achieve high performance. These jobs are independent and there is no communication requirement. This research work recognizes that high performance in Grid environment can be achieved for applications, which consist of independent jobs without having any requirements for communication among them. On the other hand, applications consisting of inter-dependent jobs generate high communication overhead, and may not be suitable for an extremely loosely coupled system like Grid.

The *Resource Broker* acts as an intermediary between the application and a set of resources. It is the responsibility of the Resource Broker to negotiate and find suitable resources according to the application's resource requirement. The *Job Controller* is responsible for controlling the execution of each of the concurrent jobs (components or tasks of an application) at local level. Besides, it also maintains a global view regarding the

application's runtime behavior, as well as the functioning of the infrastructure. It also decides run-time actions for improving the performance whenever an SLA is violated. The *Analyzer* component monitors individual resources and gathers performance-monitoring data regarding application execution, as well as infrastructure functioning. The *Performance Tuner* is responsible for tuning the performance of the application at the local level. It receives the agreement from the job controller and takes necessary actions for improving the performance whenever the agreement is violated.

These four components work cooperatively to provide the facility of *resource brokering, regular monitoring of job performance,* and *adaptive execution of concurrent jobs*, which have been identified as the major goals of the framework. Next three subsections briefly describe these functionalities.

3.2.1 Resource Brokering

As mentioned earlier, in this research work a compute-intensive Grid application, when submitted by a client, is decomposed into a collection of work items, each of which is termed as a job. These jobs form a batch and the client is allowed to characterize the requirements of each job in a batch. The resource broker first gathers resource information and produces a list of prospective resource providers for each job in the batch. Then it selects the best set of resource providers for the entire batch with the aspiration of overall time or cost optimization, i.e. aiming at minimizing the resource utilization cost (or time) for the entire batch and thereby controlling any kind of over provisioning. Initial resource allocation is performed on the basis of this selection. The resource broker establishes an SLA between the selected resource providers and the client. These jobs need to be executed onto the resources in such a way that the QoS (as promised in the SLA) is maintained. If the performance of the job is not satisfactory then the resource broker reschedules the suffered job to other suitable resource provider and also establishes a new SLA with the newly selected resource provider. Novel strategies for pursuing the above goals are incorporated within this framework.

3.2.2 Regular Monitoring

In order to maintain the QoS, the performance of the infrastructure and the jobs are monitored. Performance of the infrastructure has an effect on the performance of the job. Thus, to maintain the QoS, as well as to improve the performance of the job, performance data for both are gathered and analyzed. Different researchers focus on novel strategies for performance monitoring and analysis [39, 82, 86, 87, 95]. The focus of this thesis is particularly on resource brokering and adaptive execution (discussed later). Performance monitoring and analysis has not been brought within the scope of this research work. Therefore, as the subsequent part of the thesis will reveal, the implementation of the above framework depends on existing tools for performance monitoring and analysis.

3.2.3 Adaptive Execution

The framework provides an adaptive execution environment, which is used for achieving high performance of applications running on a Grid. The performance of a job can be improved by adapting it to the dynamic availability and capacity of Grid resources. The job controller component keeps track all the jobs in the batch executing on different resources. Whenever the job controller is notified by the analyzer regarding some performance problem on a particular resource provider, it takes appropriate action so that performance of the job does not degrade. Sometimes performance of a job may be improved by applying some local optimization techniques. Adaptation is also achieved by supporting automatic migration (rescheduling) of a job suffering performance degradation at the current node. In case of migration, a new resource provider is selected for executing the job and a new agreement is established with this new resource provider.

3.3 Conclusion

This chapter sets forth the requirements for performance-based resource management in Grid environment. Detailed study of some existing resource management systems for Grid along with discussions on their strengths and limitations in the context of execution of complex applications for high performance is put forward. This chapter also introduces a brief overview of the design of a framework for performance based resource management

in Grid. Three vital activities to be performed within the framework are identified as resource brokering, performance monitoring and adaptive execution. Based on this framework, the thesis proposes a multi-agent system that deploys a set of agents for all interactions within and among the components. The multi-agent system consists of both stationary and mobile agents. A detailed study on software agents and the existing agent-based systems has been made to design and implement this framework.

The next chapter gives an overview of software agents and also distinguishes between stationary agents and mobile agents. In addition, Chapter 4 also emphasizes on Java as a platform for implementing software agents. Finally, it discusses the detailed design of the multi-agent based resource management system using a firm software engineering approach.

Chapter 4

Multi-Agent Based Resource Management Framework

The previous chapter introduces a resource management framework for a heterogeneous, dynamic and distributed environment like Grid. The present chapter discusses implementation of this framework as a multi-agent system. The main advantage of an agent-based system is that it offers greater flexibility, dynamism and adaptability. Agent-based approaches allow splitting the entire task into a group of tasks and create groups of autonomous agents in order to commit these tasks in a collaborative manner. Each agent is intended to perform some predetermined role and is explicitly responsible for all aspects of that role. The chapter makes use of a software engineering approach for building agent-based system in order to describe how the diverse objectives of the resource management framework map properly to a multi-agent system.

The chapter first presents a brief discussion on the capabilities of software agents and subsequently, distinguishes between stationary agents and mobile agents. In particular, mobile agents form a major part of the implementation of multi-agent-based resource management system. Hence, the chapter gives additional emphasis on mobile agents and thoroughly discusses their characteristics and capabilities. It also describes why Java is a good choice for agent implementation. Finally, the chapter concentrates on the design of the multi-agent framework on the basis of a recognized software engineering approach,

namely Gaia methodology [117]. In order to evaluate the effectiveness of the multi-agent framework, a tool, PRAGMA (**P**erformance based **R**esource Brokering and **A**daptive execution in **G**rid for **M**ultiple concurrent **A**pplications) has been developed. This chapter also presents an overview of the tool.

4.1 Overview of Software Agents

A software agent has a spectrum of definitions. Some popular definitions for agents are:
"*An **agent** is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors*" [94].

*"An **autonomous agent** is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future"* [36].

"***Autonomous agents** are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed*" [65].

"***Intelligent agents** are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires*" [62].

Software agents are kind of software programs that exist to perform a specific kind of work and automate given assignments. The essential feature of software agents is autonomy. Software agents must be able to act on behalf of some other party, be it a person or another agent. To carry out the task effectively, some degree of autonomy is required. Hence, agents must be able to take action when necessary without human interaction. An agent is created in a system space by another agent or a non-agent program. In a space of agents, each agent can possess a greater or lesser degree attributes such as [8]:

- **Reactivity:** Agents recognizes the circumstances in which they operate and respond to changes in that environment appropriately.
- **Persistence:** Agent code is not executed on demand but runs continuously and decides for itself when it should perform some activity.
- **Autonomy:** Agents have the capabilities of self-starting behavior, task selection, prioritization, decision-making without human intervention and goal directed behavior.
- **Collaborative behavior:** Agents are able to engage other agents or components through some sort of communication and coordination to achieve a common objective.
- **Adaptability:** Adaptation implies sensing the environment and reconfiguring in response. This can be achieved through the choice of alternative problem solving rules or algorithms, or through the discovery of problem solving strategies. Agents have the ability to learn and they can improve their performance from the experience.
- **Mobility:** Agents can move from one host to another to accomplish task on behalf of its creator.

Software agents are suitable for use in a wide variety of applications. They can be useful to model complex tasks. Agents can be categorized on the basis of the nature of the task performed by them, which includes but not limited to:

- **Intelligent agents:** Intelligent agents are software agents that assist users and perform assigned tasks on the basis of fixed pre-programmed rules. Intelligent agents have the ability to adapt and learn.
- **Autonomous agents:** Autonomous agents are software agents that are capable of making independent decisions and taking actions to satisfy internal goals based upon their perceived environment.
- **Distributed agents:** Agents are well suited for use in applications that involve distributed computation or communication between components. Distributed agents

are very loosely coupled and they can be executed as independent threads on distributed processors.

- **Multi-agent systems:** Several agents may form a multi-agent system. All these agents work collaboratively to achieve a common objective. Characteristically such agents will not have all data or all methods to achieve that objective and thus will have to collaborate with other agents. Agents in a multi-agent system may have numerous objectives and goals, and although they are individual and autonomous they still often have to collaborate and communicate with other agents in order to achieve their objectives.

Generally, on the basis of the working places of the software agents, they can be broadly subdivided into two types: *Stationary agents* and *Mobile agents*. The working places of stationary agents are limited to the system space where they are created. Stationary agents stay on a single machine and provide a specific service to either the user or other agents. In contrast, mobile agents are not limited to the system space in which they are created; rather they are free to travel among all possible hosts in a network. While moving from one host to another, a mobile agent takes its execution state with it. Mobile agents should be able to execute on every host in a network and the agent code should not have to be installed on every host the agent could visit. There is no difference between a mobile agent and a stationary agent from the system's point of view, except that a stationary agent typically has authority to access more system resources for obvious security reasons.

Mobile agents can interact with the stationary agents according to the system requirements. In general, mobile agents obtain necessary services through requests to trusted stationary agents that perform sensitive tasks with careful security checks. Stationary agents are generally used for management purposes, whereas mobile agents are used to distribute the task among the computers in the network. Therefore, mobile agents may use Java virtual machine where classes can be loaded at runtime over the network. Mobile agents come in a variety of flavors and perform numerous functions like: an *information agent* which searches for information residing on remote nodes and report back to the source from where

the mobile agent originates; a *computational agent* which seeks underutilized network resources to perform cpu-intensive processing functions; and a *communication agent* which couriers messages back and forth between clients residing on various network nodes [13].

4.1.1 Mobile Agents

Adaptive execution of multiple concurrent jobs on the basis of real-time performance analysis is a major service, which is offered by the resource management framework. Its implementation is entirely based on *mobile agents*. Therefore, this section concentrates on mobile agents and highlights their features and benefits.

A mobile agent is an autonomous program that can move from one node to another in a network. Most important features of a mobile agent include:

- **Autonomy:** This means goal-directedness, proactive and self-starting behaviour. A mobile agent is a separate computational unit, with its own thread of execution, independent of others.
- **Mobility/Migration:** Mobile agents are able to migrate in a self-directed way from one host platform to another. This is the most important characteristic of mobile agents. There are two basic models of migration: the *weak* and the *strong* migration. Weak migration relates to transfer of only the code and data of an agent. The agent restarts on the new host from the beginning but with its data. The agent must be prepared for the transfer so that all the necessary information is embedded in the data. The strong migration transfers the execution state of the agent along with the code and data and the agent restarts from the point where it has stopped. Two popular mechanisms of migration between the nodes are used by the agents, these are remote procedure call (RPC) and sockets.
- **Communication:** A mobile agent needs to communicate with the agent platform, as well as it may want to interact with other stationary agents or mobile agents. Many communication mechanisms are used in mobile agent systems. The most used techniques are:

- **Procedure Call mechanism:** An entity A calls another entity B to perform a service during which the entity A is blocked. It is a synchronous process and difficult to parallelise.
- **Callback mechanism:** An entity A calls an entity B to perform a service but continues with its task. The entity B, when finishes its assigned task, calls back the entity A and sends the result. It permits truly asynchronous processing, but is complicated.
- **Mailbox mechanism:** An entity A calls entity B for a service and asks it to put the results into predetermined mailbox. The entity A continues its task while periodically checking its mailbox. This is also asynchronous, but more difficult to implement. [52]

For the above features, use of mobile agents has numerous attractive benefits including

- **Bandwidth Savings:** Mobile agents save the network bandwidth and communication costs by moving the computation task to the location where the data is available rather than transferring the huge amount of data.
- **Potentials for reducing network traffic:** Agents are smaller in size and sending an agent avoids repeated remote interactions. Avoiding remote interactions reduces network traffic, as well as facilitates faster completion times. A mobile agent uses the network only for a relatively short period during its migration. It is therefore capable of dealing with slow and unreliable network links.
- **Disconnected operations:** Processing may be done at remote server node, thus enabling the execution of an application on remote node while they are disconnected from the backbone network.
- **Dynamic deployment:** Code may be downloaded on the fly.
- **Load Balancing:** A task may migrate from a heavily loaded node to a relatively free node utilizing mobility of an agent.

For the above characteristics and benefits of mobile agents, they are useful as well as powerful paradigms for building distributed applications. However, they raise several

privacy and security concerns, which clearly are main obstacles to the widespread use and adaptation of this new technology. Mobile agent security issues include authentications, identification, secure messaging, certification and resource control. Mobile agent frameworks must be able to counter new threats as agent hosts must be protected from malicious agents, agents must be protected from malicious hosts, and agents must be protected from malicious agents. However, these issues are not within the scope of this thesis and therefore, have not been dealt with.

Generally, a mobile agent is a long time running process. In the course of its life span it goes through different states of its lifecycle as shown in Figure 4.1 [52]. A mobile agent starts from its *creation state.* In this initial state the mobile agent is ready to run and when the execution starts it moves to the *starting state*. When a mobile agent arrives at a new host, it will always be restored to the same state i.e., in *starting state* before it continues with the execution again. When a mobile agent needs to move to some other host, it enters in the *deactivation state*, stops all its work and saves its state and intermediate data. In a Java implementation, the execution states of the mobile agent objects are exported to a byte stream using object serialization and later they are reconstructed from the byte stream using deserialization. At last, when the mobile agent stops executing, it enters in the *disposal state*. After that, its execution states are lost forever.

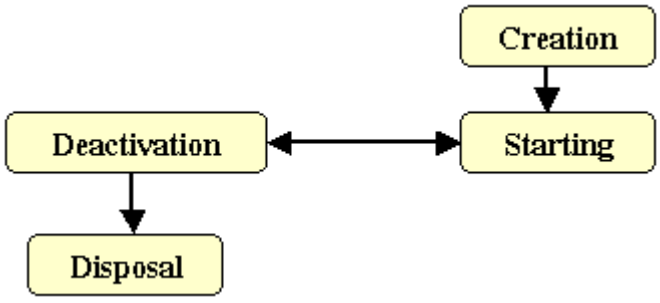


Figure 4.1: Lifecycle of a mobile agent [52].

4.1.2 Java – A Programming Paradigm for Software Agents

Java has many built-in features that meet the criteria of agent programming. Java applications are able to be executed anywhere, on any computer in a network irrespective of the underlying operating system or hardware [116]. The most useful aspects of Java are:

- **Platform Independence:** The Java platform is an ideal base for agent technology as it facilitates the platform independence of the code. This feature is especially convenient for software agents, particularly for mobile agents. A mobile agent extends the ability of an agent by removing the constraint of location and moves across a prescribed network. This means that a common platform must exist on the hosts for the agent to run. As long as there is a Java Virtual Machine (JVM) interpreting byte code and offering a mobile agent platform, the agent may run.
- **Dynamic Class Loading:** Java supports dynamic loading and linking of the code by means of a hierarchy of class loaders. A class loader constitutes a separate name space that can be used to isolate classes of the agent system and of different agents from each other.
- **Multithreaded programming:** In general, agent platforms execute multiple agents and service components or other modules concurrently. Java caters for this need by means of multithreaded programming. The Java language includes a set of APIs and primitives to manage multiple concurrent threads within a Java program.
- **Object Serialization:** While developing agent applications, this feature can be used for supporting mobility. An object (an agent) may be serialized (converted to a stream of bytes), and moved (passed over the socket) to another host where it continues its execution.
- **Remote Method Invocation (RMI):** It provides a means by which agents may communicate with each other. Information interchange between the agent platform (server) and the agents (clients) is also supported by this mechanism. It uses a distributed object application where Java objects may be accessed and their methods may be called remotely to take advantage of a distributed environment and thus spread a workload over a number of network nodes.

The conceptual model of Java-based mobile agent system is shown in Figure 4.2 [99]. The agent platform provides an agent API with basic functionalities for agent management, migration and communication in the form of Java packages. TCP/IP is used as the main transport mechanism.

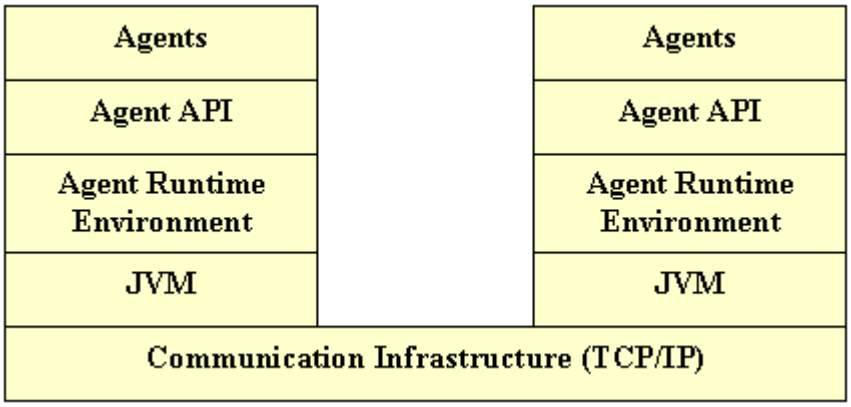


Figure 4.2: Java based mobile agent system [99].

4.2 Design of a Multi-Agent System

For the notable advantages of agent systems as revealed in the last section, the thesis proposes a multi-agent system on the basis of the resource management framework introduced in Section 3.2.

Within a multi-agent system (MAS) many intelligent agents interact with each other. Their interactions can be either cooperative or selfish, i.e., the agents can share a common goal (e.g. an ant colony), or they can pursue their own interests (as in the free market economy). A multi-agent system aims at creating a system that interconnects separately developed agents, thus enabling the group to function beyond the capabilities of any particular agent in the set up. Communication, collaboration, and coordination are important layers of interaction among these agents. Communication allows agents within the system to share information. Collaboration allows agents to mutually update shared set of information, actions and decisions. Coordination ensures the collaborative actions of the individual

agents working on a shared set of decisions are coordinated to achieve the desired objective efficiently.

The remaining part of this chapter concentrates on the design of the multi-agent system based on a software engineering approach. Within the system, a group of interacting, autonomous agents work together towards a common goal – "*to set up and maintain the service level agreement between a client of computational services and the owners of the computational resources"*. The goal is achieved by regular monitoring of the application performance and the infrastructure performance and scheduling and rescheduling jobs onto Grid resources depending upon their requirements. Each agent plays a specific role and has a well-defined set of responsibilities or subgoals in the context of the overall system and is responsible for pursuing these autonomously.

The next section explains some terminologies and outlines a methodology, namely Gaia methodology. The detailed steps for modeling the multi-agent system (MAS) using this methodology are described in the following sections.

4.2.1 The Gaia Methodology

This section presents an overview of the Gaia methodology [117], which has been used in this thesis for describing the proposed MAS for performance-based resource management. Gaia provides support to systematically move from a statement of requirements to a design that is sufficiently detailed and can be implemented directly. The main concepts in Gaia can be divided into two categories: *abstract* and *concrete*. These concepts are developed in three different phases – the analysis phase, the architectural design phase and the detailed design phase. Abstract entities are developed at the time of analyzing the requirements of the system and conceptualizing it, these entities may not have any direct realization within the system. The abstract concepts in Gaia include *roles, permissions, responsibilities, protocols, activities, liveness properties* and *safety properties*. Concrete entities, in contrast, are used in the detailed design phase, and will typically have direct counterparts in the run-

time system. Thus the *agent types, services* and *acquaintances* are developed as concrete concepts.

While analyzing a system, it is always required to identify some characteristics, in other words some basic skills which are essential for achieving the defined goal of the system. These basic skills can be attributed to different *roles* within the system. It is also necessary to understand the specific interaction patterns among these roles. Thus, Gaia proposes to develop *role* and *interaction models* for the MAS in the analysis and architectural design phases by following a number of steps. The *role model* identifies the key roles in the system and provides an abstract description of the entities' expected functions. The dependencies and relationships between various roles are represented in Gaia in the form of *interaction model*. This model consists of a set of protocol definitions, one for each type of inter-role interaction. The analysis phase and the architectural design phase for proposed multi-agent system are described in Section 4.2.2 and Section 4.2.3 respectively. In the following paragraphs some related terminologies are explained.

A *role* is described by its responsibilities, permissions, activities, and protocols. *Responsibilities* determine functionality. Responsibilities are further divided into two types: *liveness properties* and *safety properties*. Liveness properties describe those states of affairs that an agent must bring about, given certain environmental conditions. In contrast, safety properties define acceptable states of affairs that are maintained across all states of execution. In Gaia, liveness properties are specified via a *liveness expression*, which defines the "life-cycle" of the role. The general form of a liveness expression is: ROLENAME=expression. Table 4.1 shows the operators for liveness expressions. In order to carry out the responsibilities, a role has a set of *permissions*. Permissions are the "rights" associated with a role. The permissions of a role thus identify the resources that are available to that role in order to realize its responsibilities. The *activities* of a role are computations associated with the role that may be carried out by the agent without interacting with other agents. A role is also associated with a number of *protocols*, which define the way it can interact with other roles as mentioned above.

The role and interaction models obtained using the above concepts are later transformed into sufficiently low level of abstractions so that the traditional design techniques may be applied in order to implement agents. Thus, in the detailed design phase, three models are generally used. The *agent model* identifies the agent types that will make up the system, and the agent instances that will be instantiated from these types. The *services model* identifies the main services that are required to understand the agents' role. Lastly, the *acquaintance model* documents the lines of communication between different agents. Section 4.2.4 describes the models of detailed design phase in the context of the proposed multi-agent system.

Operator	**Interpretation**
X.Y	X followed by Y
X | Y	X or Y occurs
X*	X occurs 0 or more times
X+	X occurs 1 or more times
X^w	X occurs indefinitely often
[X]	X is optional
X || Y	X and Y interleaved

Table 4.1: Operators for liveness expressions [117].

4.2.2 Analysis Phase

The objective of the analysis phase is to develop an understanding of the system and its structure. In this section, first the organizations within the system are identified, and then an environment model is developed as proposed by Gaia.

Organizations

According to Gaia, the first step in the analysis phase of a MAS is to determine whether multiple organizations have to coexist in the system and perform autonomously. Such sub-

organizations can be found when there are segments of the overall system that fulfill any of these conditions:

i. Exhibit a behavior specifically oriented towards the achievement of a given sub-goal.
ii. Interact loosely with other segments of the system.
iii. Require competences that are not needed in other parts of the system.

It may be noted that there is no requirement for fulfilling all the above conditions by the sub-organizations. As pointed out before, the sub-organizations must fulfill at least one of the above conditions. In order to identify the sub-organizations, it is first necessary to recognize the goals and subgoals of the proposed MAS. The four major subgoals of the system, as noted in Chapter 3, can be defined as follows:

- resource brokering,
- controlling the execution of a single job or more than one concurrent jobs,
- performance analysis of the infrastructure and individual jobs, and
- providing adaptive execution facility to suffered job (performance tuning of a specific job at local level or rescheduling)

On the basis of this preliminary analysis, the MAS consists of four sub-organizations. The responsibilities of these four sub-organizations are (1) finding suitable resources for multiple concurrent jobs as per their resource requirements, (2) controlling the concurrent execution of jobs, and keeping the resource utilization cost of the entire batch as minimum as possible, (3) performance monitoring of the jobs and the infrastructure, and (4) improving performance at run-time in case of any performance problems. These organizations fulfill the first and the third conditions mentioned above and also partly the second condition as well. The goal of each sub-organization is shown in Table 4.2.

The Environment Model

As discussed in [120], the environment model is described as a collection of resources and active components. Resources are available to the agents for reading their values or

changing their values or extracting them from the environment. On the other hand, active components are components and services capable of performing complex operations with which agents in the MAS have to interact. The resources and the active components of the proposed MAS are identified in Table 4.3.

Name	Description
Resource Broker	This sub-organization ensures that every job gets suitable resources according to its requirements. It also guarantees the overall cost optimization of resource utilization of the entire batch of jobs.
Job Controller	This sub-organization controls concurrent execution of jobs. It maintains a global view regarding the application's runtime behavior, as well as functioning of the infrastructure. It also decides run-time actions (adaptive execution facility) for improving the performance whenever an SLA is violated.
Analyzer	This sub-organization monitors individual resources and collects performance monitoring data of job execution and underlying infrastructure functioning.
Performance Tuner	This sub-organization is responsible for tuning the performance of the application at local level.

Table 4.2: Sub-organizations in the multi-agent system.

In the next step of the analysis phase, it is realized how these resources are accessed by the sub-organizations. Based on this understanding a preliminary role model and a preliminary interaction model are developed. The objective of developing the preliminary role model is to identify the basic skills, i.e. functionalities and competences, required by a sub-organization to achieve its goals. The objective of the preliminary interaction model is to capture the dependencies and relationships between the various roles in the multi-agent system organization. The preliminary role model and preliminary interaction model are shown in Appendix A. The roles identified in this model are as follows:

i. A ResourceBroker and multiple ResourceProviders[3] in the Resource Broker sub-organization.
ii. Multiple JobControllers in the Job Controller sub-organization (one for each job).

[3] A number of components, organizations, roles and agents related terms are used in the thesis. In order to distinguish among the concepts different notations have been used. For example to indicate a resource provider component or organization the term *Resource Provider* is used, whereas, for an agent and role the term *ResourceProvider* is used.

iii. Multiple Analyzers in the Analyzer sub-organization (one associated with each resource provider).

iv. Multiple PerformanceTuners in the Performance Tuner sub-organization (one associated with each resource provider).

Next the organizational rules of the above mentioned sub-organizations are defined. Organizational rules are seen as responsibilities of the organization as a whole. As in the role model, organizational rules also have safety and liveness rules. Liveness expressions detail properties related with the dynamics of the organization, i.e. how the execution must evolve and safety expressions detail properties that must always be true during the whole life of the MAS. The organizational rules are shown in Tables 4.4 and 4.5. They have been used and considered in the architectural design phase, which will be described in the next section.

Type	**Name**	**Description**
Active Component	Client	Submits a batch of jobs with specific QoS requirements.
Active Component	Resource Provider	Provides resources for the execution of jobs.
Active Component	Information Service	Provides resource information.
Active Component	Scheduling Service	Provides job-scheduling facility.
Active Component	Monitoring Service	Monitors the performance of jobs and the infrastructure.
Resource	JRL	Contains description of each job (JDesc) and recognizes the minimal requirements for executing the job.
Resource	RST	Collection of description of all Resources.
Resource	ResourceProviderList	List of eligible Resource Providers for a job.
Resource	JobResourceMatrix	List of all the eligible Resource Providers for all jobs.
Resource	CostMatrix	Entries in the cost matrix are the estimated cost of resource usage.
Resource	JobMap	Mapping jobs onto the most suitable Resource Provider.
Resource	SLA	Agreement between client and resource provider.
Resource	PerfData	Information regarding the performance of job and infrastructure functioning.
Resource	AnalysisReport	Information regarding the performance improvement strategies.

Table 4.3: Resources and active components.

Liveness Rules (relations)	Description
CheckResourceAvailability(ResourceBroker(batch_of_jobs(x))) → *SetUpSLA(JobController(batch_of_jobs(x)))*	Protocol *CheckResourceAvailability* must necessarily be executed by role *ResourceBroker* for a specific batch of jobs before *JobController* can execute protocol *SetUpSLA* for that batch of jobs.
TransefrSLA(Analyzer(job(x))) → *RaiseWarning(Analyzer (job(x)))*	Protocol *TransferSLA* must necessarily be executed by role *Analyzer* for a specific job before *Analyzer* can execute protocol *RaiseWarning* for that job.
RaiseWarning(Analyzer (job(x))) → *AwaitCall(PerformanceTuner(job(x)))*	Protocol *RaiseWarning* must necessarily be executed by role *Analyzer* for a specific job before *PerformanceTuner* can execute protocol *AwaitCall* for that job.

Table 4.4: Liveness (relations) organizational rules.

Safety Rules (constraints)	Description
¬*(ResourceBroker* | *ResourceProvider)*	Role *ResourceBroker* and role *ResourceProvider* can never be played concurrently by the same entity.
¬*(ResourceProvider* | *JobController)*	Role *ResourceProvider* and role *JobController* can never be played concurrently by the same entity.
¬*(JobController* | *Analyzer)*	Role *JobController* and role *Analyzer* can never be played concurrently by the same entity.
¬*(JobController* | *PerformanceTuner)*	Role *JobController* and role *PerformanceTuner* can never be played concurrently by the same entity.

Table 4.5: Safety (constraints) organizational rules.

4.2.3 Architectural Design Phase

According to Gaia methodology, the basic understanding on a MAS is developed by analyzing the organization and viewing it as a collection of roles, which take part in systematic institutionalized patterns of interactions with other roles. The analysis phase has already been described in the previous section, and the preliminary role model, preliminary interaction model and the organizational rules have been developed. On the basis of the analysis, this section describes the architectural design of the MAS that completes and refines the preliminary models and makes actual decisions about the organizational structure and models the MAS based on the specifications produced. Tables 4.6

demonstrate the organizational structure of the sub-organizations within the MAS. The requirement for Job Controller organization is to control the execution of each job in a batch, while maintaining a global view of the system. Therefore, a hierarchical topology of the organization is suggested, in which one role will be recognized as a supervisory role and others are subordinates to it. According to the preliminary role model, the topologies of the other organizations are as given in Table 4.6.

The tasks of the preliminary roles are further divided into subtasks and new roles ResourceManager (in Resource Broker organization) and JobExecutionManager, SLANegotiator and SLAManager (in Job Controller organization) are introduced in this phase. The final role model is given in this section and the responsibilities of all these roles are explained. Table 4.7, 4.8, 4.9 and 4.10 show the organization structures of the four sub-organizations along with the new roles.

Organization	**Topology**	**Control regime**
Resource Broker	Hierarchy	Work specialization and work partitioning
Job Controller	Hierarchy	Work specialization and work partitioning
Analyzer	Collection of peers	Work specialization
Performance Tuner	Collection of peers	Work specialization

Table 4.6: Summary of topologies and control regimes used.

Statement/Comment
ResourceManager $\xrightarrow{depends\_on}$ *ResourceBroker*
Role *ResourceManager* relies on JRLs received from role *ResourceBroker*.
$\forall i$,*ResourceManager* $\xrightarrow{depends\_on}$ *ResourceProvider[i]*
Role *ResourceManager* relies on resource information from role *ResourceProvider*.
ResourceBroker $\xrightarrow{depends\_on}$ *ResourceManager*
Role *ResourceBroker* relies on ResourceProviderList from role *ResourceManager*.

Table 4.7: Organization structure for Resource Broker sub-organization.

Statement/Comment
JobController $\xrightarrow{depends\_on}$ *SLANegotiator*
Role *JobController* receives final SLA from role *SLANegotiator*.
SLANegotiator $\xrightarrow{peer}$ *SLAManager*
Role *SLANegotiator* and *SLAManager* are peers and they collaborate to finalize SLA .
JobController $\xrightarrow{control}$ *JobExecutionManager*
Role *JobController* has an authoritative relationship with role *JobExecutionManager*.
JobExecutionManager $\xrightarrow{depends\_on}$ *SLANegotiator*
Role *JobExecutionManager* receives final SLA from role *SLANegotiator*.

Table 4.8: Organization structure for Job Controller sub-organization.

Statement/Comment
Analyzer $\xrightarrow{depends\_on}$ *JobExecutionManager*
Role *Analyzer* relies on SLA from role *JobExecutionManager*.
JobExecutionManager $\xrightarrow{depends\_on}$ *Analyzer*
Role *JobExecutionManager* relies on the performance analysis report from role *Analyzer*.

Table 4.9: Organization structure for Analyzer sub-organization.

Statement/Comment
PerformanceTuner $\xrightarrow{depends\_on}$ *JobExecutionManager*
Role *PerformanceTuner* relies on resources or knowledge from role *JobExecutionManager*.

Table 4.10: Organization structure for Performance Tuner sub-organization.

Next the agent role model of the MAS is described on the basis of the analysis and organizational structure. The role schemas are presented using the templates from [117, 120]. The first role is a ResourceBroker (RB), which bears the responsibility of finding suitable resources for multiple jobs submitted to the system by one or more clients. The ResourceBroker prepares a Job Requirement List (JRL) from the description of each job and consults the ResourceManager (RM) to find suitable resources for execution of these

jobs. A ResourceManager role is responsible for maintaining the information about the resources and their quality and matching this information with the job requirement. The ResourceProvider (RP) at a particular Grid Site prepares a ResourceSpecificationMemo specifying the resources that are available for the clients. This memo not only describes the available resources at a particular Grid site, but also specifies the administrative policies within that domain. The ResourceSpecificationMemo is stored in a ResourceSpecificationTable (RST) until the RP withdraws all of its services. RM matches the JRL with the ResourceSpecificationMemos in RST and a match response along with a ResourceProviderList is sent to the ResourceBroker. A ResourceProviderList is a collection of all the resource providers who can meet the requirements of a particular job. The collection of all ResourceProviderLists for all the concurrent jobs forms a JobResourceMatrix. Figure 4.3 represents the ResourceBroker, the ResourceManager and the ResourceProvider roles.

A JobController (JC) is responsible for establishing *Service Level Agreements (SLA)* between the client and the resource owners. It decides the optimal mapping of all the concurrent jobs onto the Grid sites and accordingly creates a JobMap. Thus, while the JobResourceMatrix maintains all the resource providers who can fulfill the requirements of each job, the JobMap shows a likely scheduling of jobs onto the available resources based on a novel algorithm (described in the next chapter) which aims at minimizing the overall cost of executing the jobs. Service Level Agreements are established between the client and the resource owner at a particular Grid site through a SLAManager and a SLANegotiator. SLANegotiator negotiates with the SLAManagers on behalf of the client. A SLAManager is responsible for negotiating on behalf of the resource owner in a particular Grid site. SLANegotiator relies on the JobMap in order to decide whom to talk first. However, in case the SLAManager disagrees to finalize an SLA (it may happen if the resource owner withdraws its services for some reason), the SLANegotiator may talk to other SLAManagers (on the basis of the entries in JobResourceMatrix). Whenever a SLAManager accepts the SLA, the SLANegotiator finalizes the agreement, updates the JobMap, if necessary, and returns the final version of the SLA to the JobController.

Figure 4.4 shows these three roles, namely JobController, SLAManager and SLANegotiator.

While JobController maintains the JobMap and keeps track of the final SLAs for each of the concurrent jobs submitted by a client, a JobExecutionManager (JEM) becomes associated with each job and controls and keeps track of its execution. Thus a JobExecutionManager liaises with the Analyzer (explained later) and as soon as it receives warning from the Analyzer regarding an SLA violation (or over-provisioning) for which rescheduling is necessary, it gets back to the SLANegotiator and obtains an SLA which is an agreement with another resource owner (selected from the JobResourceMatrix) and reschedules the job there. The protocol SetUpNewSLA is thus similar to the protocol SetUpSLA, only with the exception that in this case the JEM is the initiator instead of JC. Sometimes the Analyzer may indicate that the performance of the application can be improved by applying some local optimization techniques, such as load balancing. The WarningMessage generated by the Analyzer holds this indication and on the basis of this indication the JEM either reschedules or invokes a PerformanceTuner sitting on the particular Grid site.

The JobExecutionManager role is depicted in Figure 4.5 and the Analyzer and the PerformanceTuner are shown in Figure 4.6. Although detailed description of the activities carried out by the Analyzer and the PerformanceTuner has not been brought within the scope of this thesis.

Role Schema: **ResourceBroker(RB)**

Description:
This role involves preparing a JobResourceMatrix that holds the information about all the Grid sites, which can fulfill the requirement of each of the submitted jobs.

Protocols and Activities:
PrepareJRL, CheckResourceAvailability, PrepareJobResourceMatrix

Permissions:		
	Reads	**JDesc, ResourceProviderList**
	Generates	**JRL, JobResourceMatrix**

Responsibilities
Liveness:
ResourceBroker = (PrepareJRL , CheckResourceAvailability)^n . PrepareJobResourceMatrix
Safety:
Number of Jobs = Number of Rows in JobResourceMatrix

Role Schema: **ResourceManager (RM)**

Description: This role involves matching the job requirement with the resource capabilities of the resources offering services at any point of time.

Protocols and Activities:
MatchRequest, MatchJRL, ReturnMatchResponse

Permissions:

Reads	**JRL, ResourceSpecificationTable**	
Generates	**ResourceMatchesJRL**	**// true or false**
	ResourceProviderList	

Responsibilities
Liveness**:**
ResourceManager=MatchRequest .MatchJRL . ReturnMatchResponse
Safety:
Number of JRLs received = Number of MatchResponses returned

Role Schema: **ResourceProvider (RP)**

Description:
This role involves preparing ResourceSpecificationMemo, which describes the resources and their quality.

Protocols and Activities:
PrepareResourceSpecification, RegisterResourceProvider, WithdrawResourceProvider

Permissions:

Generates	**ResourceSpecificationMemo**	
Changes	**ResourceRegistered**	**// true or false**
	ResourceSpecificationTable	

Responsibilities
Liveness**:**
ResourceProvider = (PrepareResourceSpecification . RegisterResourceProvider) | WithdrawResourceProvider
Safety:
true

Figure 4.3: Role schemas for ResourceBroker, ResourceManager and ResourceProvider.

Role Schema: **JobController (JC)**

Description: This role involves creating and maintaining the JobMap and establishing SLAs between the clients and the Resource Providers.

Protocols and Activities:

CreateJobMap, PrepareSLA, SetUpSLA, InvokeJobExecutionManager

Permissions:

Reads	**JobResourceMatrix, JRL**	
Generates	**SLAs, SLARequests**	**// for all the concurrent jobs**
Changes	**JobMap**	

Responsibilities

Liveness**:**

JobController = CreateJobMap. (PrepareSLA . SetUpSLA. InvokeJobExecutionManager) ^{n}

Safety:

Number of entries in JobMap = Number of SLAs and SLARequests

Role Schema: **SLANegotiator (SLA_N)**

Description:
This role negotiates an SLA on behalf of the client.

Protocols and Activities:

RequestSLASignature, FinalizeSLA

Permissions:

Reads	**JobResourceMatrix, SLAResponse**
Changes	**JobMap**

Responsibilities

Liveness:

SLANegotiator = (RequestSLASignature)$^{+}$**. FinalizeSLA**

Safety:

SLAResponse = true

Role Schema: **SLAManager (SLA_M)**

Description:
This role manages the SLAs at the resource owner side.

Protocols and Activities:

SLARequest, AcceptSLA, RejectSLA, SendResponse

Permissions:

Reads	**SLA**
Generates	**SLAResponse**

Responsibilities

Liveness:

SLAManager = SLARequest. (AcceptSLA | RejectSLA)$^{+}$**. SendResponse**

Safety:

true

Figure 4.4: Role schemas for JobController, SLANegotiator and SLAManager.

Role Schema: **JobExecutionManager (JEM)**

Description: This role involves submission of job, sending SLA to analyzer, receiving warning from analyzer, rescheduling the job on the basis of some warning.

Protocols and Activities:
AwaitWarning, SubmitJob, SetUpNewSLA, ProcessJob

Permissions:
Reads **SLA, WarningMessage**

Responsibilities
Liveness**:**
JobExecutionManager = SubmitJob . ((AwaitWarning)^{w} . Reschedule | InitiateTuning)^{w} || ProcessJob^{w}
Reschedule = (SetUpNewSLA . SubmitJob)

Safety:
true

Figure 4.5: Role schema of JobExecutionManager.

Role Schema: **Analyzer**

Description: This role involves analyzing the performance of job and environment on the basis of the finalized SLA.

Protocols and Activities:
TransferSLA, MonitorJob, Analyze, RaiseWarning

Permissions:
Reads **SLA**
Generates **WarningMessage**

Responsibilities
Liveness**:**
Analyzer = TransferSLA . ((MonitorJob . Analyze)^{w} . RaiseWarning)^{w}

Safety:
true

Role Schema: **PerformanceTuner**

Description: This role involves performance tuning of a job.

Protocols and Activities:
AwaitCall, Tune

Permissions:
Reads **WarningMessage**

Responsibilities
Liveness**:**
PerformanceTuner = ((AwaitCall)^{w} . Tune)^{w}
Safety:
true

Figure 4.6: Role schemas of Analyzer and PerformanceTuner.

The roles interact with each other through the protocols as depicted in Figure 4.7, 4.8, 4.9 and 4.10. The major protocols, namely CheckResourceAvailability, SetUpSLA, RequestSLASignature and AwaitWarning are shown in these figures along with the initiators, participants, inputs and outputs of each protocol.

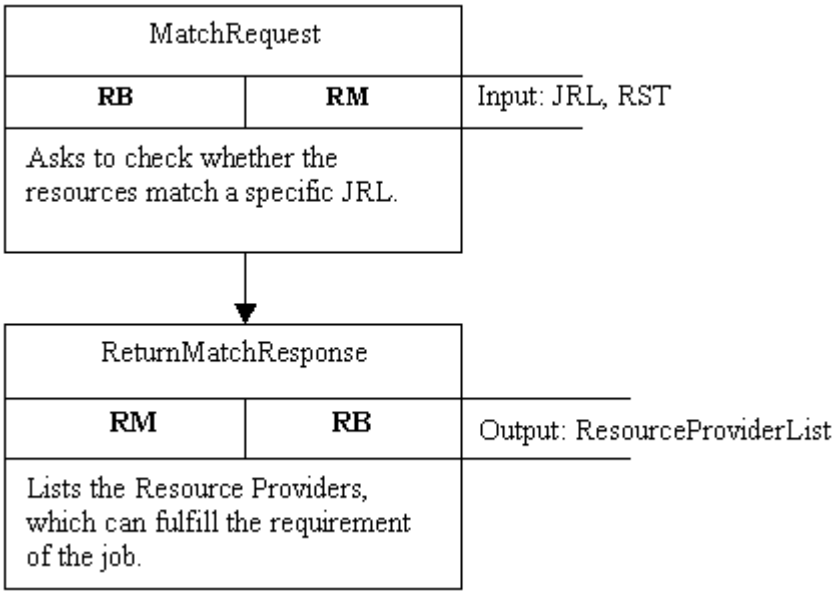


Figure 4.7: Interaction model of protocol CheckResourceAvailability.

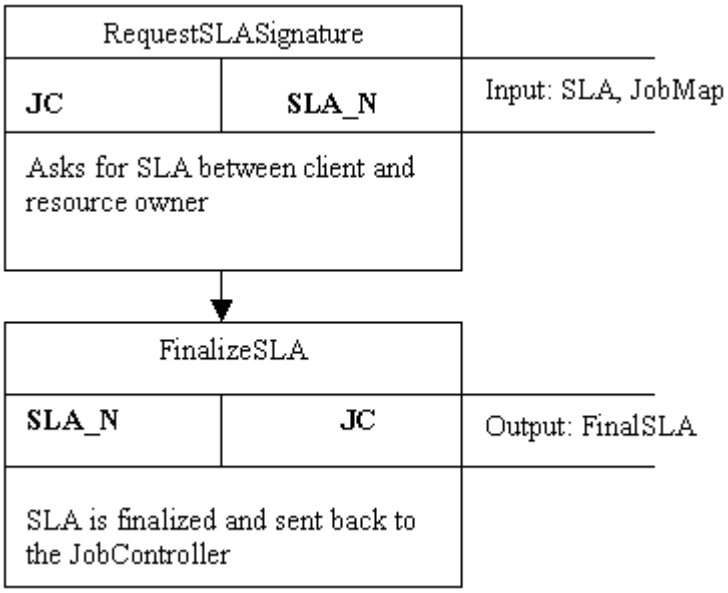


Figure 4.8: Interaction model of protocol SetUpSLA.

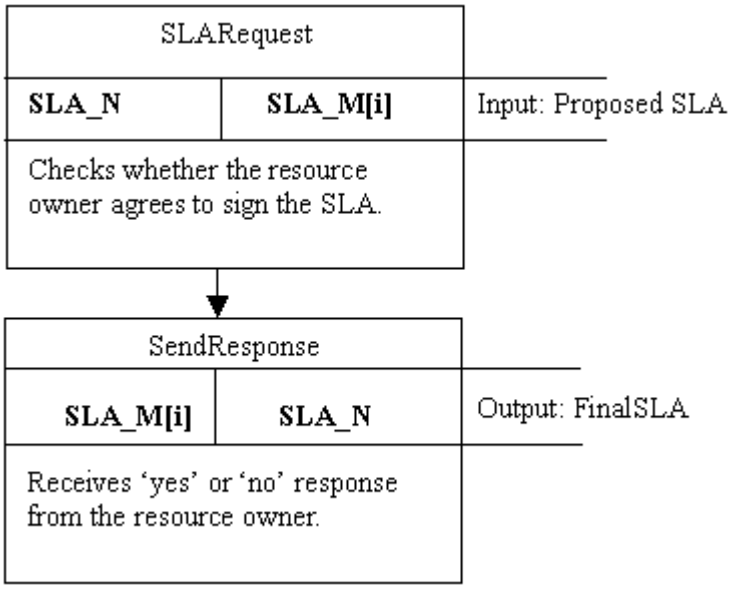


Figure 4.9: Interaction model of protocol RequestSLASignature.

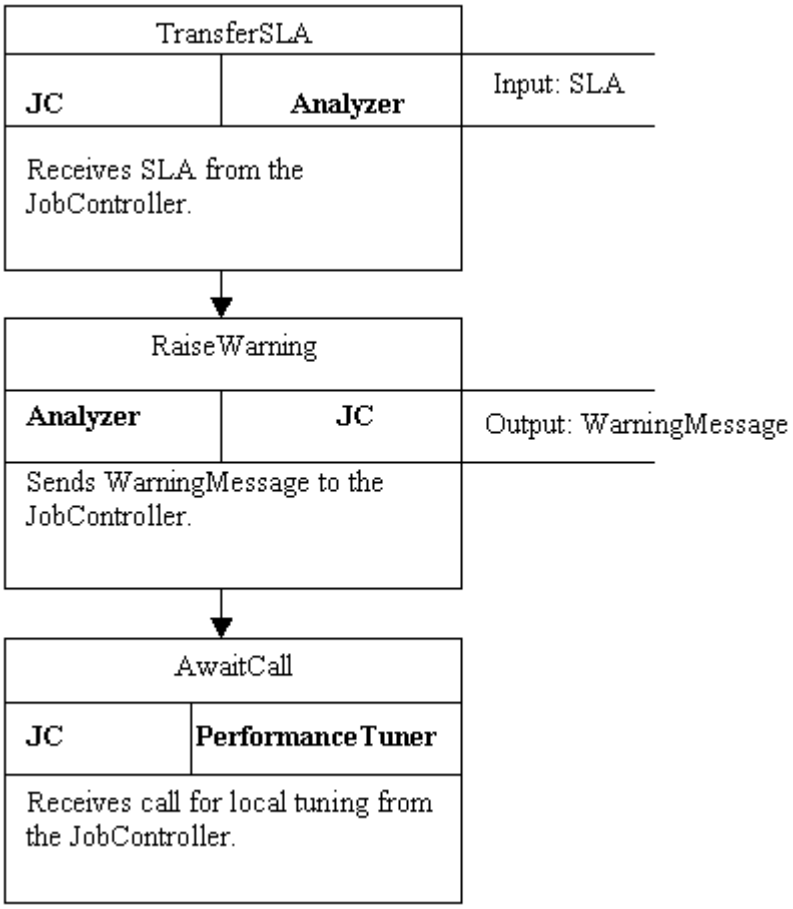


Figure 4.10: Interaction model of protocol AwaitWarning.

4.2.4 Detailed Design Phase

In the detailed design phase of Gaia methodology, an *agent model* along with the *services model* and the *acquaintance model* are developed. For the agent model, a one-to-one correspondence can be made between roles and agent classes. However, there are some advantages in trying to compact the design by reducing the number of classes and instances leading to a reduction in conceptual complexity. It also needs to be pointed out that an agent footprint (in terms of memory and processing power) can be large and in order to increase the efficiency of the system, it is always preferable to limit the number of agents. However, this has to be done without:

- affecting the organizational efficiency,
- violating organizational rules and
- Creating – "bounded rationality" problems (that is, without exceeding the amount of information it is able to store and process in a given amount of time). [16]

With the above considerations, it can be noted that the tasks of ResourceBroker and ResourceManager can be accomplished by one single agent. An agent, which owns the JRL, can collect the RST from the ResourceProviders and perform the matching. Similarly, the JobController, which creates the JobMap, can also negotiate for setting up an SLA and thereby carry out the tasks of the SLANegotiator. The tasks of a SLAManager can be handled by an agent, which plays the role of a ResourceProvider. This type of reductions in number of agents will not upset the coherency of the system, but will help in increasing its efficiency.

A mapping between the roles and the corresponding agents in the proposed agent model is presented in Table 4.11. The numbers within parentheses in the right-hand side column indicates the number of agents of each type in the system. This thesis will mainly concentrate on the implementation of the first four agents. Therefore, in the table, the other two are shown in different colours.

Role	Agent
ResourceBroker (RB)	Broker Agent (1)
ResourceManager (RM)	
ResourceProvider (RP)	ResourceProvider Agent (M)
SLAManager (SLA_M)	
JobController (JC)	JobController Agent (1)
SLANegotiator (SLA_N)	
JobExecutionManager (JEM)	JobExecutionManager Agent (N)
Analyzer	Analysis Agent (N)
PerformanceTuner	Tuning Agent (N)

Table 4.11: Mapping between roles and agents.

As the table demonstrates, altogether six agents are introduced in the multi-agent system. These are:

- Broker Agent,
- ResourceProvider Agent,
- JobController Agent,
- JobExecutionManager Agent,
- Analysis Agent, and
- Tuning Agent.

Among these six agents, the Analysis and Tuning agents play single roles, i.e. the roles of Analyzer and the PerformanceTuner. JobExecutionManager also plays a single role. However, the other three, namely the Broker Agent, the ResourceProvider Agent and the JobController Agent play multiple roles.

The services model identifies the services that each agent will offer. This may be derived from the protocols, activities and liveness properties of the roles that the agent implements as described in the previous section. As a rule of thumb, there will be one service for each parallel activity of execution that the agent has to execute. However, it might be possible to

introduce more services even for sequential activities of execution, especially in the cases where it is necessary to represent different phases of the execution. In this case, for each activity, a service is developed on the basis of appropriate algorithms. Also, for each of the protocols described in the last section, a corresponding service is implemented. The service model requires that, for each service that may be performed by an agent, four properties are identified: inputs, outputs, pre-conditions and post-conditions. In case of services that are obtained from the interaction model (involve agent interaction or exchange of knowledge between agents), the inputs and outputs are derived from the protocols. For the services corresponding to the activities involving evaluation and modification of the environment resources, the inputs and outputs come from the environment. The pre and post conditions represent restrictions on the execution and completion of the services. These can be derived from the role safety properties as well as from the organizational rules. Table 4.12, 4.13, 4.14, 4.15, 4.16 and 4.17 shows the service model of all agents.

Service	**Input**	**Output**	**Pre-condition**	**Post-condition**
Job Modeling	JDesc	JRL	Submission of job	A successful creation of JRL
Resource Selection	JRL, RST, JobResourceMatrix, CostMatrix	JobMap	Successful connection with Resource Providers	A successful creation of JobMap

Table 4.12: Service model for Broker Agent.

Service	**Input**	**Output**	**Pre-condition**	**Post-condition**
Resource Discovery	ResourceSpecificationMemo	RST	Successful connection with Information Service	A successful creation of RST

Table 4.13: Service model for ResourceProvider Agent.

Service	Input	Output	Pre-condition	Post-condition
Resource Selection at the time of initial scheduling	JobResourceMatrix, CostMatrix	JobMap		A successful creation of JobMap
SLA Setup	JRL, ResourceSpecificationMemo	SLA	Successful connection with Client and ResourceProvider	A successful creation of SLA
Initial Scheduling	JobMap, SLA		SLAs for all jobs have been set up.	Successful start of job execution

Table 4.14: Service model for JobController Agent.

Service	Input	Output	Pre-condition	Post-condition
Resource Selection at the time of Rescheduling	ResourceProviderList, CostVector	JobMap	Successful connection with Resource Providers and Information Service	A successful creation of JobMap
New SLA Setup	JRL, ResourceSpecificationMemo	SLA	Successful connection with Client and ResourceProvider	A successful creation of new SLA
Rescheduling	JobMap, SLA		SLA has been set up.	Successful restart of job execution

Table 4.15: Service model for JobExecutionManager Agent.

Service	Input	Output	Pre-condition	Post-condition
Collects performance monitoring data	SLA	PerfData	Successful connection with Monitoring and Information Service	
Analyze Performance data	PerfData	AnalysisReport		A Successful creation of Analysis Report

Table 4.16: Service model for Analysis Agent.

Service	Input	Output	Pre-condition	Post-condition
Perform local tuning	SLA, AnalysisReport	Tuning Actions	Receives warning from JEM	

Table 4.17: Service model for Tuning Agent.

All the agents in the system have equally important roles to play and there is no question of holding a master-slave relationship. However, the JobController Agent supervises all the JobExecutionManager Agents, which are associated with the multiple concurrent jobs (one JEM for each job). Thus, the JobExecutionManager Agents become subordinates to the JobController Agent and work in close collaboration with the latter. Figure 4.11 demonstrates the interactions among the agents at the time of initial scheduling of a single job onto the Grid resources. Figure 4.12 demonstrates the situation of rescheduling, where JobExecutionManager Agent collects updated RST from the ResourceProvider Agent and selects the next resource provider for the job. Figure 4.11 and Figure 4.12 shows that the agents also need to interact with some external services, such as scheduling service, monitoring service and information service, which are provided by some standard middleware and tools.

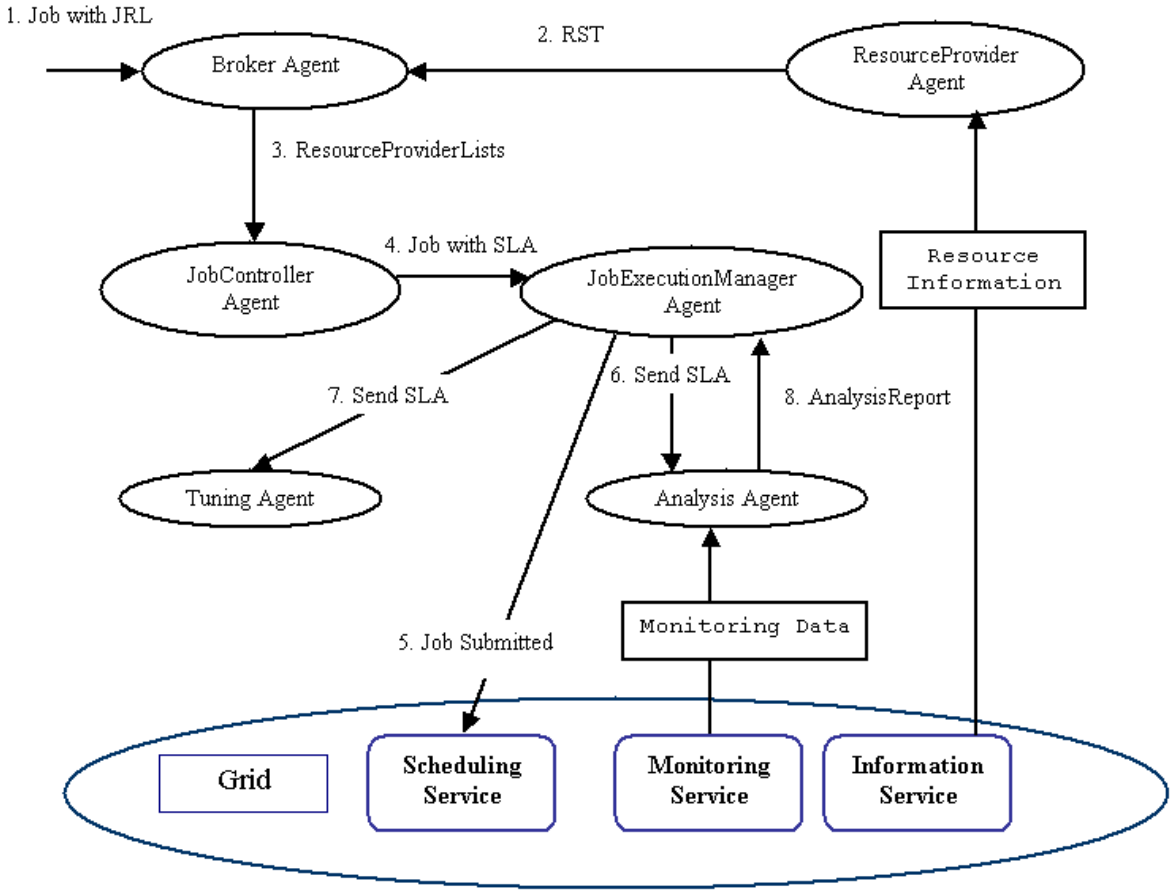


Figure 4.11: Agent interactions at the time of scheduling.

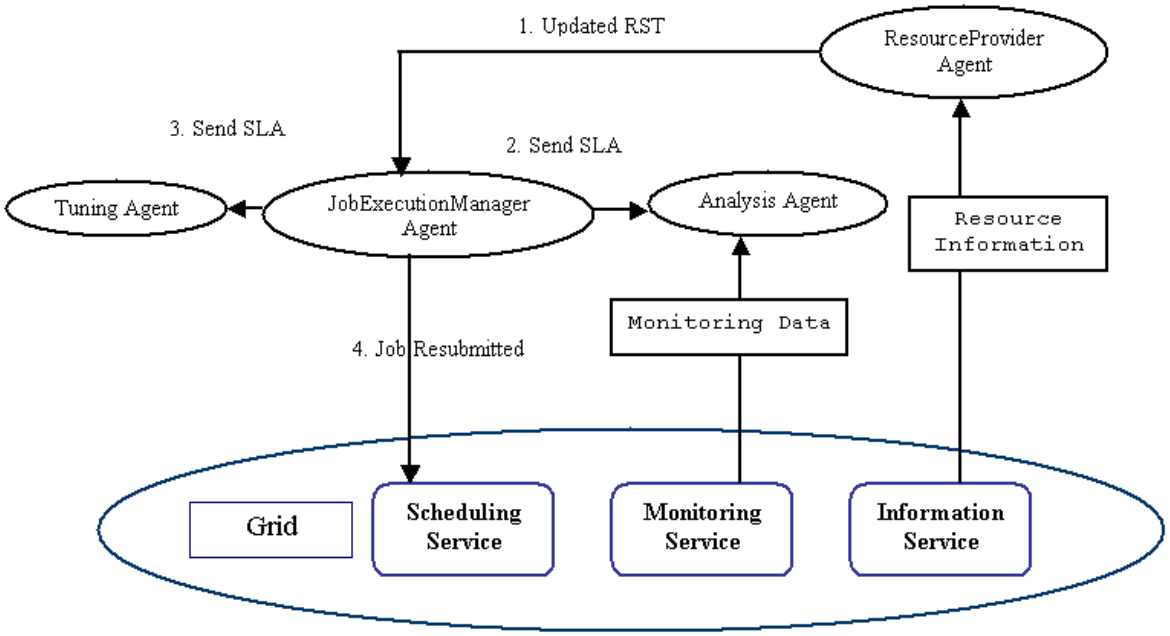


Figure 4.12: Agent interactions at the time of rescheduling.

A tool, PRAGMA (**P**erformance based **R**esource Brokering and **A**daptive execution in **G**rid for **M**ultiple concurrent **A**pplications) has been developed on the basis of the design presented in this section. The tool is deployed on top of the most used Grid middleware Globus Toolkit 4 and adds the desired resource management functionalities to the basic components of the middleware. A brief description of the tool is given in the next section.

4.3 Overview of PRAGMA

The major objectives of PRAGMA are *resource brokering* and *adaptive execution* that may improve the application performance. Figure 4.13 depicts an overview of PRAGMA, which consists of four components (as discussed in Chapter 3). Every component consists of one or two agents as shown in this figure. These agents can communicate with each other within the component or outside the component, if required. The Resource Broker component along with the Job Controller component performs the resource brokering for a batch of jobs with the objective of overall cost optimization. Whenever a user submits jobs to the Grid, PRAGMA first finds suitable resources for their execution, sets up service-level agreements (SLA) between the resource owners and the Grid user. The jobs are then

scheduled onto the selected resources. At the time of execution, if it is found that any resource is unable to fulfill the agreement, mobile agents (handled by the JEMs) are employed for migrating the job to another resource (upon the suggestion of the resource broker). All the decisions are taken on the basis of regular monitoring and analysis of execution performance of the application, as well as the infrastructure.

PRAGMA handles a batch of jobs and searches suitable resource providers for all jobs to provide the execution environment. While selecting the resources, PRAGMA aims at optimizing the overall execution time or resource usage cost for execution of the entire batch. This is a unique feature of PRAGMA, which is not present in most of the existing Grid resource broker system. The major tasks of the four components of PRAGMA include:

- **Resource Broker**: Job modeling, Resource discovery.
- **Job Controller:** Resource selection, Resource allocation, SLA setup and maintenance, Resource selection at the time of adaptive execution.
- **Analyzer:** Performance analysis of jobs and execution environment.
- **Performance Tuner:** Local performance tuning.

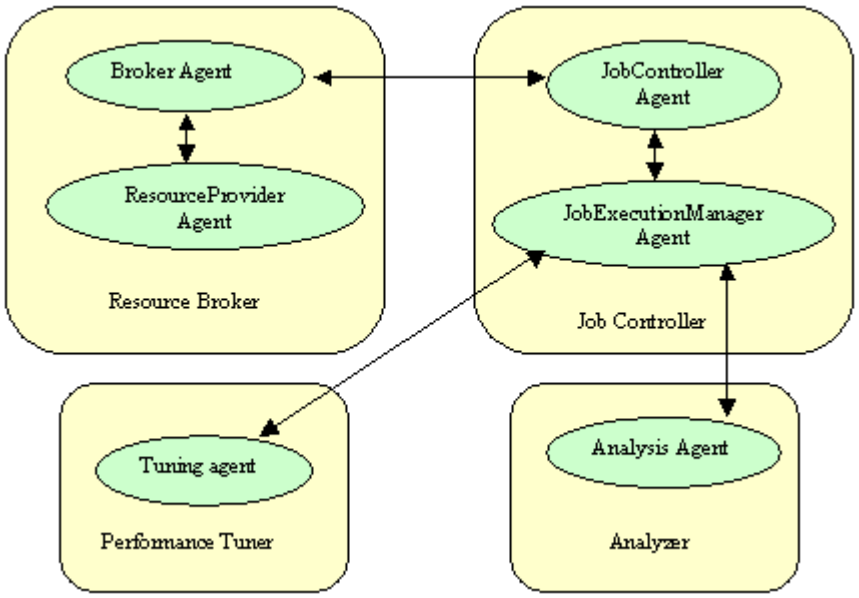


Figure 4.13: Four components with agents.

The Broker Agent and the ResourceProvider Agent play key roles in resource brokering. Initially the *Broker Agent* and the *ResourceProvider Agent* perform the task of resource brokering. Later, the *JobController Agent* takes over the responsibility of run-time resource-allocation on the basis of the changing resource requirements and resource usage scenarios and dynamically reschedules the jobs as and when required. The adaptive execution facility is managed by the Job Controller component along with the Analyzer component and the Performance Tuner component. An overall agent class diagram is shown in Figure 4.14.

Figure 4.14: Agent class diagram.

4.4 Conclusion

This chapter presents the current scenario in agent technology and elaborates on the design of a multi-agent system for performance-based resource management in Grid environment. A software engineering approach based on Gaia Methodology is used to evolve the design

of the multi-agent system in a systematic manner. The Role model and Interaction model in the architectural design phase and the Agent Model and Services Model in the design phase are developed following the approach. The multi-agent system is designed to provide services like resource brokering, regular monitoring (and analysis of monitoring data) of jobs and the environment, and adaptive execution (on the basis of monitoring information). This thesis pays its main attention to resource brokering and adaptive execution of the applications running in Grid environment. Four among the six agents of the proposed multi-agent system, namely Broker Agent, ResourceProvider Agent, JobController Agent and the JobExecutionManager Agent are employed for carrying out the services related to resource brokering and adaptive execution. Although, the Broker Agent initiates the resource brokering activity by creating an initial job-resource map for the entire batch of jobs, resource brokering is not entirely centralized in the system. The ResourceProvider Agents and the JobController Agent also take part in initial scheduling and later, the JobExecutionManager Agents associated with each job assist in resource brokering in a distributed manner. The hybrid approach increases the efficiency of the system (as evident in the next chapter). The next chapter will discuss how this multi-agent framework provides the resource brokering services. Chapter 5 will introduce novel algorithms for resource selection for a batch of jobs. Implementation detail of resource brokering component in PRAGMA will also be discussed in Chapter 5.

Chapter 5

Resource Brokering Strategies

It has already been discussed in the previous chapters that managing resources in Grid is an intricate issue and is not possible at the application or user level. Grid provides uniform access to heterogeneous resources owned by multiple organizations. Any time a new resource can join the Grid or an existing resource can withdraw. The system loads and status of the resources also change frequently. Thus, a *resource broker* is an essential component in a Grid environment that can assist in the selection of a right resource provider for a job in all aspects.

This chapter discusses the design and implementation of resource brokering strategies within the multi-agent framework (presented in the last chapter). These strategies help in finding an optimal allocation of resources for executing multiple concurrent jobs in a Grid environment. Section 5.1 summarises the requirements for a Grid resource broker. A novel resource selection algorithm for initial allocation of resources is discussed in this chapter. The algorithm deals with a batch of jobs and selects suitable resource provider for each job. The algorithm is based on two initial steps - (i) job modeling and (ii) resource discovery; these two steps are described in the Sections 5.2 and 5.3 respectively. Section 5.4 explains the initial resource selection algorithm. Section 5.5 discusses the approach to resource selection at the time of rescheduling. After initial resource selection, the resource broker prepares and sets up an SLA between the client and the resource provider of each job.

Section 5.6 presents the structure of the SLAs. Finally, the design and implementation of various phases of resource brokering by PRAGMA are described in Section 5.7.

5.1 Grid Resource Broker

A resource broker is a central component in a Grid environment. The main function of a Grid resource broker is to dynamically identify and characterize the available resources, and to select and allocate the most appropriate resources for a given job. A Grid resource broker must have complete information regarding jobs and the available resources. Thus, availability of complete and up to date information is a fundamental requirement for a Grid resource broker.

Maintaining quality of service (QoS) is another important task of a Grid resource broker. Depending on the job type, the client can specify quality of service (QoS) requirements. The QoS is maintained through *service level agreements*, which are managed by the Grid resource broker. Grid resource broker handles the setting up and maintenance of service level agreements (SLA) between the clients and the resource providers and maintains QoS on the basis of regular monitoring of job performance and the infrastructure.

Recent research on Grid resource brokers mainly revolves around two different types - centralized and distributed. A centralized broker manages and schedules all jobs submitted to the Grid, whereas a distributed broker typically handles jobs submitted by a single user only. Centralized brokers are able to produce optimal schedules as they have full knowledge of the jobs and resources, but such a broker can face a performance bottleneck problem. Distributed broker architecture, scales well and makes the Grid more fault-tolerant, but the partial information available to each instance of the broker complicates the scheduling decisions. For better efficiency, a hybrid type of hierarchical broker may be used in which centralized broker schedules distributed brokers. Grid resource management systems employing centralized broker include Condor [20] and EDG [26]. Distributed brokers are implemented by AppLeS [6] and NetSolve [73].

This thesis proposes a hybrid approach for resource brokering. Resource brokering for the purpose of initial scheduling of a batch of jobs is centralized, whereas resource brokering for rescheduling purpose is performed in distributed manner. The case of rescheduling occurs only when performance of a job suffers on a particular resource provider and the SLA is violated.

As pointed out in Chapter 3, resource brokering for multiple concurrent jobs with the ambition of overall time / cost optimization are not particularly handled by the existing resource brokers. Thus, this thesis focuses on evolving novel strategies for resource brokering of multiple concurrent jobs. It has been assumed that the jobs have no communication requirements. Dealing with communication requirements among the jobs will certainly increase the complexity of the algorithm.

The requirements of a Grid resource broker as identified in this thesis are summarized as below:

- Characterizing the requirements of each job.
- Providing dynamic up-to-date information regarding the available resources.
- Managing multiple organizational policies as resources may participate from multiple administrative domains.
- Selecting appropriate resources for the entire batch.
- Optimizing the overall execution time or resource usage cost for a batch of jobs.
- Achieving best performance on the basis of the promised QoS.

In general, Grid resource brokering involves three main phases [119]:

- **Job Modeling:** At this stage either the users are allowed to characterize the requirements of a job or the requirements are automatically formulated from the description of the job.
- **Resource Discovery:** This phase involves gathering resource information (up to date) and producing a list of prospective resources.

- **Resource Selection:** At this stage, the best set of resource providers for the entire batch of jobs is selected.

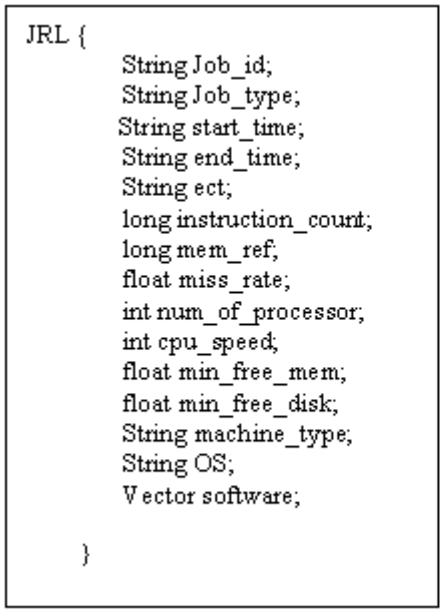
The thesis considers a Grid application as a collection of work items or *jobs* that carry out a complex computing task by using Grid resources. The set $A=\{J_1, J_2, \ldots, J_n\}$ is the batch of jobs (or components of an application). The set $GR=\{R_1, R_2, \ldots, R_m\}$ defines the collection of available resource providers in the Grid environment. The next three sections concentrate on finding an optimal mapping for the set A onto the set GR through the three stages of resource brokering.

5.2 Job Modeling

The first stage of resource brokering involves determining the types of resources that are required for a job. Requirements for each job must be identified which include number of processors, memory requirement, operating system and software requirements. The more details are incorporated, the better the matching effort can be made. The Broker Agent prepares a Job Requirement List (JRL) from the description of each job (JDesc) and identifies the minimal requirement to run that job for meeting the desired QoS. This JRL is later used to select resource providers, which can fulfill the requirements of the job. In order to achieve this objective, a suitable structure must be defined for the JRL so that important characteristics of the job are properly identified and minimal requirements for execution of the job are defined. Thus the data model defining the job requirements is represented by the JRL structure as shown in Figure 5.1 along with their meanings.

The values of the entries are obtained either through pre-execution analysis of the job or through historical data. For example, total number of instructions may be obtained using a source code analyzer. It should be noted that the client should be able to provide the probable values for the input parameters in order to obtain an approximate value for the instruction count. Similarly, client may get the miss rate using cache simulators [14] and it is specified in the corresponding JRL. Cache simulators take an address trace of the instruction and data references, simulate the cache, generate hit and miss data. Client can

suggest the cache properties, e.g., cache size, block size, associativity, and block replacement strategy in the simulator and observe the consequences for the miss rate.



Job_id: A unique identification number assigned by the Broker Agent.
Job_type: Type of the job i.e., memory bound or CPU bound (computational).
start_time: Preferred start time for the execution of the job.
end_time: Preferred end time for the execution of the job.
ect: Expected completion time of the job.
instruction_count: Total number of instructions in the program.
mem_ref: Number of memory references.
miss_rate: Miss rate in specified condition.
num_of_processor: Minimum number of processors required.
cpu_speed: Minimum speed of the cpu required
min_free_mem: Minimum free memory required.
min_free_disk: Minimum free disk space required.
machine_type: Required machine type
OS: Operating System.
software: Required softwares, if any.

Figure 5.1: Structure of JRL.

5.3 Resource Discovery

The next stage of resource brokering is to choose a set of resource providers that can fulfill the requirements of the submitted jobs. Policies maintained by the Grid organizations play an important role here [23]. The first step of resource discovery is to collect dynamic information about the available resource providers in the Grid in order to make the best possible match between jobs and resource providers. The system gathers the information to determine whether a resource provider for a particular job is available and the present status of the available resources. Resource status of a resource provider changes, so it is very important to gather all this information in real time. Information providers, like in other distributed environments, play an important role in resource brokering in Grid. The resource broker takes decisions on the basis of the resource information (that it gathers from the information providers) and the requirements of the clients (as specified in the JRL).

Once the information is obtained from the information providers, the *ResourceProvider Agent* prepares a ResourceSpecificationMemo specifying the resources owned by a particular host that are available for the use by the Grid clients. ResourceSpecificationMemo represents the data model used for resource discovery. The resource information is collected as metric value pairs. A ResourceSpecificationTable (RST) is built by the ResourceProvider Agent as a collection of several ResourceSpecificationMemo. The entries in the ResourceSpecificationMemo structure are shown in Figure 5.2 along with their meanings. Figure 5.3(a) shows a sample RST.

```
ResourceSpecificationMemo {

        String Resource_id;
        String Resource_type;
        String avail_day;
        String avail_time_day;
        String not_avail_time_day;
        float cost;
        File Policy;
        String architecture;
        String interconnectionnetwork;
        int num_of_processor;
        int cpu_speed;
        float peak_flops;
        int max_instn_percycle;
        int max_flops_percycle;
        float global_memory;
        float per_processor_memory;
        float cache_size;
        int mtu;
        float bytes_in;
        float bytes_out;
        float pkts_in;
        float pkts_out;
        String OS;
        Vector software;
        float hit_time;
        float miss_penalty;
        String resource_IP/URI;

        }
```

Resource_id: A unique identification number to identify the resource provider.
Resource_type: Type of the Resource Provider e.g., suitable for memory bound jobs or CPU bound jobs.
avail_day: Period of availability in a week.
avail_time_day: Period of availability in a day.
not_avail_time_day: Period of non-availability in a day.
cost: Cost of resource usage for unit amount of time.
Policy: Defined by the Grid organization.
architecture: SMP, NUMA, Cluster.
interconnectionnetwork: Bus, crossbar switch, multistage switch.
num_of_processor: Number of processors available.
cpu_speed: Speed of the cpu.
peak_flops: Theoretical Floating Point Operations.
max_instn_percycle: Maximum number of instructions per cycle.
max_flops_percycle: Maximum number of FLOPS per cycle.
global_memory: Total global memory available.
per_processor_memory: Per processor memory available.
cache_size: Available cache memory.
mtu: Message transfer unit.
bytes_in: Number of bytes in per second
bytes_out: Number of bytes out per second
pkts_in: Packets in per second
pkts_out: Packets out per second
OS: Operating System.
software: Available software
hit_time: Time to hit in the cache.
miss_penalty: Number of clock cycles penalty in case of a miss.
resource_IP/URI: IP/URI of the resource provider.

Figure 5.2: Structure of ResourceSpecificationMemo.

5.4 Resource Selection

As dynamic information about resources is available, resource selection becomes more straightforward and scalable. The entire resource selection procedure is performed in two stages. In the first stage, the resource information is used to find all the resource providers who can fulfill the requirements of a particular job. In the second part of the resource selection process, the actual allocation of resources to all the concurrent jobs is done on the basis of a resource allocation algorithm, which optimizes the time / cost of running these jobs onto resources and also ensures that every job gets allocated to a resource that can fulfill its requirement and meet the quality of service.

5.4.1 Phase 1: Building JobResourceMatrix

The objective of this phase is to generate a JobResourceMatrix, which has an entry for every job showing the resource providers that can fulfill the requirements of the job. A JRL is prepared from the description of each job (JDesc). The JRL is then matched with the entries in RST. The Broker Agent compares the JRL of job "J_i" and the ResourceSpecificationMemos of all the available resource provider R_k, $\forall k$ from "GR" and prepares a ResourceProviderList(i), i.e. the set ResourceProviderList(i) $\subseteq$ GR and the members of this set are ready to provide computational services for J_i. Thus a ResourceProviderList is a collection of all the resource providers who can meet the requirements of a particular job. The collection of all ResourceProviderLists for all the concurrent jobs forms a JobResourceMatrix. A ResourceProviderList for job J_i becomes a row in the JobResourceMatrix and the k-th cell of this row is 1 if R_k can satisfy the requirements of J_i, it is 0 otherwise. Figure 5.3(b) shows an example ResourceProviderList.

The Broker Agent compares each metric-value of the JRL of job J_i with the metrics in ResourceSpecificationMemo using some specific operations. Operations are the usual binary comparisons including EQ, NE, GE, GT, LE, LT corresponding to equality, inequality, greater than or equal, greater than, less than or equal, and less than. These operations are used in JRL to specify the formal parameters for comparison. In general, a JRL represents the minimum requirements of a job. While, operation "IS" is used by

resource provider to specify an actual value of its resources in ResourceSpecificationMemo. The comparison is made not on the basis of a simple string-matching algorithm, but on the basis of a set of rules, which are used for intelligently comparing the values. Thus, if the available resource capacity is equal to or more than the resource requirement of a job, the particular resource provider is considered for it. The Broker Agent also checks whether the resource provider is available throughout the period of time when the client job needs to be executed. For example, if some resource provider provides their resources during a specific time period, e.g., between 8am and 5pm on weekdays only and if a job needs to be executed on weekends or at a different time, then the resource provider is not considered for the particular job. The Broker Agent also handles such situation efficiently. Below is the outline of the algorithm, which executes as soon as the batch of jobs is ready.

Input: Job Descriptions from the client.
Output: ResourceProviderList, JobResourceMatrix.
Action: Prepares Job Requirement List (JRL), builds ResourceSpecificationTable (RST), matches JRL and RST and prepares JobResourceMatrix.

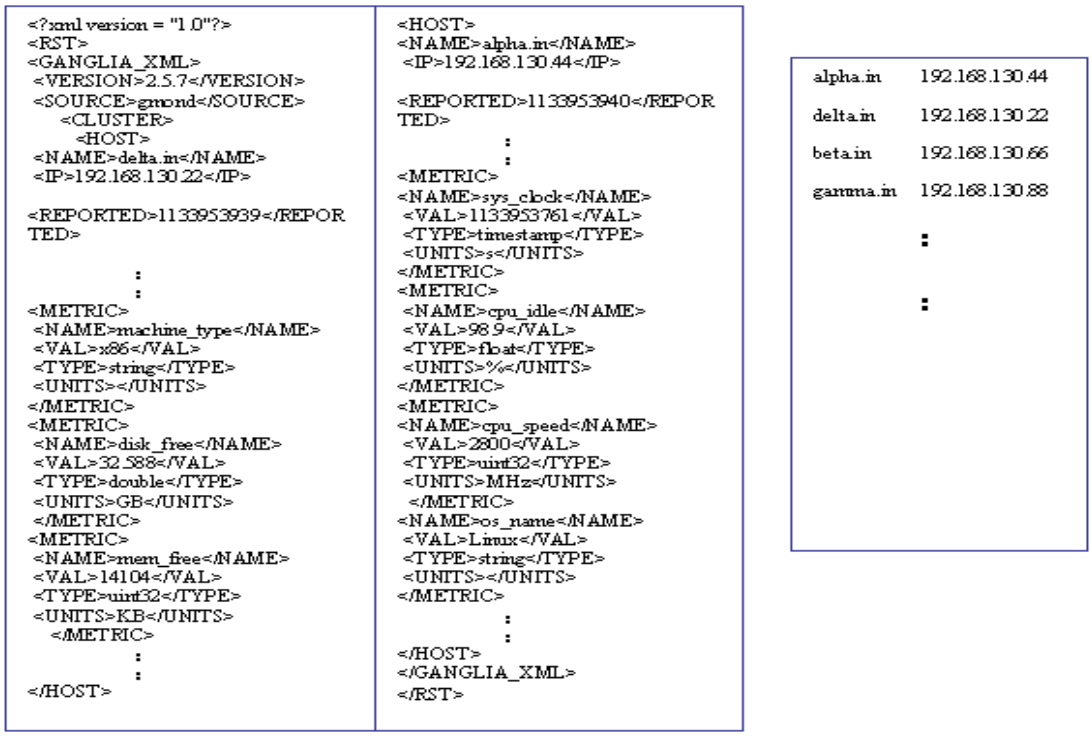
I. For all jobs J_i in batch A={J_1, J_2…J_i…} do

1) Read the JDesc (job description) and prepare the JRL for the job J_i.
2) Receive the resource information from the information provider. Prepare and store the RST in XML format from the output of information provider.
3) In order to match the JRL with RST do
   For each ResourceSpecificationMemo in the RST do
   (i) flag=0;
   (ii) For each metric α in the JRL do
   (a) matched=0;
   (b) For each metric β in ResourceSpecificationMemo where α.name= β.name do
   If α.val matched with β.val using α.op then matched=1;
   Else { flag=1; break; } //not matched
   (iii) If (flag == 1) continue; // next ResourceSpecificationMemo
   (iv) If (matched==1) then Insert the name and IP of the host in the ResourceProviderList of job J_i //successful match

II. Build the JobResourceMatrix with the collection of the ResourceProviderLists for all the jobs in batch A.
   If resource provider R_k meets the requirement of job J_i then
   JobResourceMatrix[J_i, R_k]=1;
   Else JobResourceMatrix[J_i, R_k]=0;

Time complexity of this algorithm is $\mathbf{O(N_J N_{RP})}$, where $\mathbf{N_J}$ is the number of jobs present in a batch and $\mathbf{N_{RP}}$ is the number of Resource Providers i.e., $\mathbf{N_{RP}}$ number of distinct

ResourceSpecificationMemo are present in the RST. JRLs of all job submitted in a batch are matched with $\mathbf{N_{RP}}$ number of ResourceSpecificationMemos of the RST. Figure 5.4 shows a sample JobResourceMatrix.



```
<?xml version = "1.0"?>
<RST>
<GANGLIA_XML>
 <VERSION>2.5.7</VERSION>
 <SOURCE>gmond</SOURCE>
    <CLUSTER>
      <HOST>
 <NAME>delta.in</NAME>
 <IP>192.168.130.22</IP>

<REPORTED>1133953939</REPOR
TED>

            :
            :
<METRIC>
 <NAME>machine_type</NAME>
 <VAL>x86</VAL>
 <TYPE>string</TYPE>
 <UNITS></UNITS>
</METRIC>
<METRIC>
 <NAME>disk_free</NAME>
 <VAL>32.588</VAL>
 <TYPE>double</TYPE>
 <UNITS>GB</UNITS>
 </METRIC>
<METRIC>
 <NAME>mem_free</NAME>
 <VAL>14104</VAL>
 <TYPE>uint32</TYPE>
 <UNITS>KB</UNITS>
   </METRIC>
            :
            :
</HOST>
```

```
<HOST>
<NAME>alpha.in</NAME>
 <IP>192.168.130.44</IP>

<REPORTED>1133953940</REPOR
TED>
            :
            :
<METRIC>
<NAME>sys_clock</NAME>
 <VAL>1133953761</VAL>
 <TYPE>timestamp</TYPE>
 <UNITS>s</UNITS>
</METRIC>
<METRIC>
 <NAME>cpu_idle</NAME>
 <VAL>98.9</VAL>
 <TYPE>float</TYPE>
 <UNITS>%</UNITS>
</METRIC>
<METRIC>
<NAME>cpu_speed</NAME>
 <VAL>2800</VAL>
 <TYPE>uint32</TYPE>
 <UNITS>MHz</UNITS>
  </METRIC>
<NAME>os_name</NAME>
 <VAL>Linux</VAL>
 <TYPE>string</TYPE>
 <UNITS></UNITS>
</METRIC>
            :
            :
</HOST>
</GANGLIA_XML>
</RST>
```

```
alpha.in    192.168.130.44
delta.in    192.168.130.22
beta.in     192.168.130.66
gamma.in    192.168.130.88
            :
            :
```

Figure 5.3: (a) RST. (b) ResourceProviderList.

	R_1	R_2	R_3	R_4	R_5	...	R_m
J_1	1	0	0	1	0	...	0
J_2	0	1	1	0	1	...	0
J_3	1	0	1	0	0	...	1
J_4	0	0	1	0	0	...	1
...	...	...	...	...	...	...	...
J_n	1	0	0	1	1	...	0

Figure 5.4: JobResourceMatrix.

Actual allocation of resources to all jobs in the batch is made on the basis of an optimizing strategy, which is explained in the next two subsections. The approach proposed for this purpose has two objectives, first it guarantees that the quality of service for each job is

maintained, and second the overall time for job execution or the overall resource usage cost is optimized. The objective of the resource selection algorithm is to see that every job in a batch gets allocated to a suitable resource provider. Furthermore, when more than one resource providers are found to be suitable for a job, it needs to be decided which one should be selected for executing the job. Many researchers propose to select the resource provider on which the job will be executed for minimum amount of time or on which the resource usage cost will be minimum [12]. This thesis argues that only looking for the minimum execution time / resource usage cost for a particular job will not effectively allocate the resources to every job in a batch. An overall time / cost optimization strategy is therefore required. Thus, an initial resource selection algorithm based on the Munkres' assignment [70] algorithm is proposed. Application of this strategy is the responsibility of the JobController Agent. The next section explains the techniques used for cost computation in order to decide the optimum resource allocation and the following section discusses an initial resource allocation algorithm on the basis of this strategy.

5.4.2 Phase 2: Computing Cost

The algorithm selects the resources on the basis of a cost component, which can be an estimated execution time of the job on a particular resource provider or an estimated resource usage cost on that resource provider. Therefore, in addition to the JobResourceMatrix, the JobController Agent prepares a CostMatrix with the help of the JobResourceMatrix, JRLs and RST. The entries in the CostMatrix are costs, which are used to decide an optimum resource allocation strategy.

The Broker Agent ensures that resource providers meet certain minimum requirements for a job. Resource providers that do not qualify under these criteria are not given any cost values. The CostMatrix contains actual entries for R_k if and only if for a job J_i, JobResourceMatrix[J_i,R_k]=1. Cost[J_i,R_k] is the estimated cost of using the resource R_k for the job J_i. If JobResourceMatrix[J_i,R_k]=0, a dummy large value is assigned to Cost[J_i,R_k]; the value must be larger than any other entry in the cost matrix, say α. A lower cost value indicates a better match for the job. In order to obtain this cost, an approximation of the

execution time of a job on a particular resource provider is used. For this purpose only two components are considered – the CPU time and the memory access time and other costs are considered to be negligible. The estimated CPU time for a job J_i on the resource R_k is denoted by $T_{cpu}(J_i, R_k)$ and the estimated time involved in memory access for the job J_i on the resource provider R_k is denoted by $T_{ma}(J_i, R_k)$. Cost of resource usage per unit time on a resource provider R_k is denoted by cost(R_k), which is specified in the ResourceSpecificationMemo. The total cost of using the resource provider R_k for the job J_i is thus calculated as shown below:

$$C(J_i, R_k) = (T_{cpu}(J_i, R_k) + T_{ma}(J_i, R_k)) \text{ X } cost(R_k)$$

The estimated minimum CPU time i.e., $T_{cpu}(J_i,R_k)$ for job J_i on the Resource Provider R_k is given by:

$$T_{cpu}(J_i, R_k) = I_c \text{ X } 1/I_m \text{ X } 1/clock\_rate$$

Where,

I_c: Instruction count.
I_m: Maximum number of instructions per cycle.

The instruction count of a program is specified in the JRL and maximum number of instructions per cycle and the clock rate of a resource provider are specified in the ResourceSpecificationMemo.

A measure of memory hierarchy performance is given by taking an estimation of the average memory access time. The definition of average memory access time $AMAT_{Rk}$ [51] at the different levels in the memory hierarchy of resource provider R_k is given by:

$$AMAT_{Rk} = hit\_time + miss\_rate \text{ X } miss\_penalty$$

Hit time and miss penalty are given in the ResourceSpecificationMemo of each resource provider. Assuming just one level of cache in the hierarchy and considering M_i as the number of memory references which are expected while executing the job J_i (specified in JRL), a simple estimation of the memory access time of a job J_i on the resource provider R_k is given by:

$$T_{ma}(J_i, R_k) = M_i \text{ X } AMAT_{Rk}$$

5.4.3 Phase 3: Initial Resource Selection Algorithm

The next action is to allot jobs to the resource providers in such a way that:

i. Every job in the batch gets allocated to some resource provider which can fulfill its requirement,
ii. The overall resource usage cost is optimized.

Three popular batch mode heuristics for independent job scheduling are *min-min* heuristic, *max-min* heuristic, and *suffrage* heuristic [27, 66]. The *min-min* heuristic begins with the batch A of all unmapped jobs. Then, the set of minimum completion time M for each job in A is found. The job with the overall minimum completion time from M is selected and assigned to the corresponding resource provider (hence the name *min-min*). The newly mapped job and the selected RP are removed from A and the set of RPs respectively. The process repeats until all jobs are mapped (i.e., A is empty) [66]. The *max-min* heuristic is very similar to *min-min*. It also begins with the batch A of all unmapped jobs. Then, the set of minimum completion time M is found. Next, the job with the overall maximum from M is selected and assigned to the corresponding resource provider (hence the name *max-min*). Finally, the newly mapped job and the selected RP are removed from A and the set of RPs respectively and the process repeats until all jobs are mapped (i.e., A is empty) [66]. Another popular heuristic for independent job scheduling is called *suffrage*. The rationale behind *suffrage* is that a job should be assigned to a certain host and if it does not go to that host, it will suffer the most. For each job, its suffrage value is defined as the difference between its best minimum completion time and its second-best minimum completion time. Tasks with high suffrage value take precedence [66].

All these algorithms use the notion of minimum completion time to assign a job to its suitable resource provider. The algorithm proposed in this section considers the minimum estimated cost of executing job J_i, on resource provider R_k in place of minimum completion time and also attempts to find an optimum resource mapping for all the jobs in the batch. *Max-min* algorithm is not suitable for this purpose, because it picks a job from batch A with maximum estimated cost, whereas the objective of this thesis is to optimize the overall cost.

Min-min and *suffrage* heuristics fail to give the optimized selection of resources for a given batch of jobs such that the overall cost of executing the jobs on respective resource providers is minimized. A slight modification of the algorithm proposed in this section may attempt to optimize the overall execution time of the batch of jobs instead of minimizing the overall resource usage cost.

Below the outline of the initial resource selection algorithm using the concept of Munkre's assignment algorithm (also known as Hungarian method) is provided. The Hungarian method is a combinatorial optimization algorithm, which solves assignment problems in polynomial time [70]. The assignment model is actually a special case of the transportation model in which the jobs represent the sources and the RPs represent the destinations. The algorithm models an assignment problem using the $\mathbf{N_J}$ **X** $\mathbf{N_{RP}}$ cost matrix. As mentioned earlier, $\mathbf{N_J}$ is the total number of jobs present in the batch and $\mathbf{N_{RP}}$ is the total number of resource providers available.

Input: JRL, RST, JobResourceMatrix, CostMatrix.
Output: JobMap.
Action: Prepares JobMap for all jobs in the batch A.

Step 1: If (N_J != N_{RP})

I. Convert the CostMatrix into a square matrix by adding dummy rows or columns.

II. Assign a large number (larger than any other entry in the CostMatrix, say α) to the entries of the dummy rows or columns.

III. If (N_J > N_{RP}) N=N_J
Else N= N_{RP}

Step 2: Obtain reduced cost matrix, $\mathbf{C_R}$ using the following steps:

I. For each row do
Find the minimum entry in the row and subtract the entry from every other entry in that row. After subtraction, each row has at least one zero and all other entries are greater than or equal to zero.

II. For each column do
Find the minimum entry in the column and subtract the entry from every other entry in that column. After subtraction, each column has at least one zero and all other entries are greater than or equal to zero.

Step 3: Select minimum number of rows and columns in such a way, that all zeros are covered by the selection. Let L be the total number of selected rows and columns.

Step 4: If (L< N) do the following steps:

I. Find the smallest number not covered by the selection.

II. Subtract this number from itself and every other uncovered number.

III. Add this number to the intersection of the selected rows and columns.

IV. Go to Step 3.

Step 5: If (L=N) //Find optimal solution for P jobs using the zero entries of the cost matrix.

I. Initially P=0

II. For each row in $\mathbf{C_R}$ do
If it contains a single zero at location (i,j) do the following steps:
a. If the (i,j)-th in the original cost matrix **C** is less than **α**, do
i. Allocate the i-th job to the j-th Resource Provider and add an entry in the JobMap with this information.
ii. Increment P by 1.
iii. Remove the i-th row and j-th column from $\mathbf{C_R}$.

III. For each column in $\mathbf{C_R}$ do
If it contains a single zero at location (i,j) do the following steps:
a. If the (i,j)-th in the original cost matrix **C** is less than **α**, do
i. Allocate the i-th job to the j-th Resource Provider and add an entry in the JobMap with this information.
ii. Increment P by 1.
iii. Remove the i-th row and j-th column from $\mathbf{C_R}$.

Step 6: Obtain the remaining set of jobs as N_J=N-P.

Step 7: Repeat Step 1 through Step 6 with the set of remaining jobs until all jobs are assigned to some resource provider.

An example is given below to demonstrate the functioning of the algorithm. In the example, N_J (Total number of jobs present in the batch) = 3 and N_{RP} (Total number of resource providers available) = 4. Consider an example cost matrix.

	R_1	R_2	R_3	R_4
J_1	1	2	3	4
J_2	2	4	6	8
J_3	3	6	9	12

$N_J < N_{RP}$
Convert the cost matrix into a square matrix (NXN) by adding dummy row

	R_1	R_2	R_3	R_4
J_1	1	2	3	4
J_2	2	4	6	8
J_3	3	6	9	12
Dummy Row	α	α	α	α

Now, N= N_J (including dummy job) = N_{RP} =4;

	R_1	R_2	R_3	R_4
J_1	0	1	2	3
J_2	0	2	4	6
J_3	0	3	6	9
Dummy Row	0	0	0	0

After subtracting row minimum from each row

	R_1	R_2	R_3	R_4
J_1	0	1	2	3
J_2	0	2	4	6
J_3	0	3	6	9
Dummy Row	0	0	0	0

After subtracting column minimum from each column

Select minimum number of rows and columns in such a way, that all zeros are covered by the selection. Let L be the total number of selected rows and columns.

	R_1	R_2	R_3	R_4
J_1	0	1	2	3
J_2	0	2	4	6
J_3	0	3	6	9
Dummy Row	0	0	0	0

Here L=2. i.e., L is less than N
The smallest number not covered by the selection is 1, at position (1,2).

	R_1	R_2	R_3	R_4
J_1	~~0~~	~~0~~	~~1~~	~~2~~
J_2	0	1	3	5
J_3	0	2	5	8
Dummy Row	~~1~~	~~0~~	~~0~~	~~0~~

After Subtracting 1 from other uncovered numbers, and adding it to the intersection of the selected rows and columns.
L=3 i.e., L is less than N
The smallest number not covered by the selection is 1, at position (2,2).

	R_1	R_2	R_3	R_4
J_1	~~1~~	~~0~~	1	2
J_2	0	0	2	4
J_3	0	1	4	7
Dummy Row	~~2~~	~~0~~	~~0~~	~~0~~

After Subtracting 1 from other uncovered numbers, and adding it to the intersection of the selected rows and columns.
L=3 i.e., L is less than N
The smallest number not covered by the selection is 1, at position (1,3).

	R_1	R_2	R_3	R_4
J_1	~~1~~	~~0~~	~~0~~	1
J_2	0	0	1	3
J_3	0	1	3	6
Dummy Row	~~3~~	~~1~~	~~0~~	~~0~~

After Subtracting 1 from other uncovered numbers, and adding it to the intersection of the selected rows and columns.
Now, L=4 i.e., L is equal to N

Optimal Solution for all jobs using the zero entries of the cost matrix:

	R_1	R_2	R_3	R_4
J_1	1	0	0	1
J_2	0	0	1	3
J_3	0	1	3	6
Dummy Row	3	1	0	0

Assign J_3 to R_1. Resource allocation cost is 3.
(Single zero at (3,1) location, remove 3rd row and 1st column)

	R_2	R_3	R_4
J_1	0	0	1
J_2	0	1	3
Dummy Row	1	0	0

Assign J_2 to R_2. Resource allocation cost is 4.
(Single zero at (2,1) location, remove 2nd row and 1st column)

	R_3	R_4
J_1	0	1
Dummy Row	0	0

Assign J_1 to R_3. Resource allocation cost is 3.
(Single zero at (1,1) location, remove 1st row and 1st column)

Total cost of resource allocation is: 3+4+3=10.

	R_1	R_2	R_3	R_4
J_1	1	2	3	4
J_2	2	4	6	8
J_3	3	6	9	12

An example of min-min algorithm is given below using the same example cost matrix.

	R_1	R_2	R_3	R_4
J_1	1	2	3	4
J_2	2	4	6	8
J_3	3	6	9	12

M={1,2,3}; Assign J_1 to R_1, remove corresponding row and column for J_1 and R_1.
New M is {4,6}, assign J_2 to R_2, and remove corresponding row and column J_2 and R_2.
New M is {9} assign J_3 to R_3.

Total cost of resource allocation is: 1+4+9=14.

Now the JobMap for all jobs in the batch are ready for initial scheduling. The structure of a JobMap is shown in Figure 5.5. If the number of jobs in the batch is greater than the number of available resource providers, a particular resource provider can be selected for executing more than one job as shown in the algorithm (steps 6 and 7). The JobController Agent, at this point, sends an SLA (discussed later) to each selected resource provider and upon receiving their responses submits the respective jobs onto the resource providers. As the dynamic information from all the resource providers are collected just before running the above algorithm (Resource Discovery phase), it can be assumed the resource providers will agree with the SLA at this point time. However, in case a resource provider does not agree with the SLA, the job needs to be rescheduled and the steps discussed in Section 5.5 should be followed in such occasions.

```
JobMap {

        String Job_id;
        String Resource_id;

        }
```

Figure 5.5: Structure of JobMap.

The algorithm presented in this section gives cost optimal solution for the initial scheduling of jobs. Time complexity of this algorithm is $\mathbf{O(N^3)}$, (NXN is the size of the cost matrix after converting it to a square matrix, i.e. after adding the dummy rows or columns as described in the algorithm) which is the time complexity of Hungarian method [70]. Once a job is scheduled to the selected resource provider, the Analysis Agents in the framework start monitoring the performance of the job and the infrastructure. If the performance of the job degrades, the agents take some actions in order to improve the performance. These actions include local tuning and rescheduling of the job to another resource provider. The next section discusses the resource brokering strategies at the time of rescheduling a job onto a different resource provider.

5.5 Resource Selection at the time of Rescheduling

When a performance problem occurs at the time of execution of a job, an Analysis Agent sitting on the particular host (see Chapter 7 and Chapter 8) alerts the JobExecutionManager Agent associated with the job regarding the problem. Depending upon the nature of the problem, Analysis Agent either suggests local tuning actions or reschedules the job on a different host. In case of rescheduling a job J_i, a new resource provider R_k must be selected. Grid being a dynamic environment, resource selection at the time of rescheduling cannot be done without consultation with the agents deployed for brokering services. Thus, at this stage the brokering activities are distributed among the different levels of agents. The JobExecutionManager Agent (JEM) carries the ResourceProviderList for the job J_i along with the associated cost vector (a row in the cost matrix corresponding to job J_i) (see Section 5.4.2). When a problem occurs, the JobExecutionManager Agent first selects the resource provider R_k from the ResourceProviderList with a minimum entry in the cost vector. However, before rescheduling, it also verifies that the resource provider R_k is not overloaded. This is done by sending an SLA (discussed later) to the newly selected resource provider. The process bypasses any interaction with other agents like JobController Agent, Broker Agent and ResourceProvider Agent. However, the information that the job is being rescheduled is sent to the JobController Agent and the

JobMap is updated. This first level of resource brokering also avoids complexities involved in the algorithm presented in Section 5.4.3.

In case of unavailability of such resource provider (which may happen if the resource provider disagrees to accept the SLA), JobExecutionManager Agent contacts the ResourceProvider Agent and collects updated RST. JobExecutionManager Agent prepares the ResourceProviderList and the cost vector (for a single job) using the matching algorithm and the cost computation technique discussed in Sections 5.4.1 and 5.4.2. In the above context, invoking Broker or JobController Agent may cause high overhead, because these agents perform resource brokering for a batch of independent jobs and adopt a centralized approach. Here, JobExecutionManager Agents are entrusted to perform resource brokering for a single job at this level and a distributed approach is taken. Outline of the algorithm for rescheduling is given below.

Input: ResourceProviderList, CostVector.
Output: Updated JobMap.
Action: Select a new resource provider for the suffered job.

Step 1: Select a new resource provider for job J_i from existing ResourceProviderList using the following steps:

Select a resource provider R_k with the minimum entry in the CostVector and verify the status of resource provider R_k.

a. If R_k is not overloaded then establish a new SLA and reschedule Job J_i to resource provider R_k.
b. If R_k is overloaded then continue Step 1 for remaining resource providers from the ResourceProviderList.

Step 2: If no suitable resource provider is available form the existing ResourceProviderList then collect updated RST and do the followings:

a. Prepare new ResourceProviderList and CostVector for Job J_i.
b. Select a new resource provider R_l with a minimum entry in CostVector.
c. Establish a new SLA and reschedule Job J_i to resource provider R_l.

5.6 Service Level Agreement

Once the resource provider is selected by the resource broker for a specific job, the JobController Agent initiates the process of setting up a service level agreement with that resource provider. For example, a job may require an environment with minimum 4 processors and Linux operating system from 22-04-2007:10:00:00 to 24-04-2007:10:00:00. In this example, client's expected completion time of this particular job is 40 hours, which

is specified in the JRL. The resource provider must agree to an SLA which specifies all these requirements. The concepts of TSLA, RSLA, and BSLA have been introduced in Section 2.5. Figure 5.6 shows a sample JRL in XML format that acts as a TSLA and provides the description of the job in terms of job requirements. Figure 5.7 shows a sample ResourceSpecificationMemo in XML format, which is an RSLA and describes what a resource provider promises to offer. It also contains special information like 2-3-4-5-6 to indicate weekdays. Table 2.1 of Chapter 2 shows an example structure of an SLA specification. The SLA must be established between three parties including the client, the resource provider and the resource broker. The SLO (service level objective) of an SLA ensures the availability of resources that satisfy job requirements for the duration of the job execution. SLO attributes include CPU speed, operating system, disk free, available memory, machine type etc. The value of each SLO attribute is its service level indicator (SLI). As discusses in Section 2.5, two levels of quality of service (QoS) are used, namely guaranteed-service and controlled load. In a "Guaranteed-service" QoS, the SLA consists of exact constraint that must be fulfilled by the resource provider whereas in a "Controlled-load" QoS, the QoS requirements are more relaxed. "Guaranteed-service" QoS is also known as hard-QoS and it enforce the strict maintenance of the exact constraint; this is the reason why it is a better option than "Controlled-load" QoS.

After creating the JobMap, the JobController Agent establishes an agreement with the resource provider on behalf of the client. If the resource provider accepts the agreement, it guarantees to provide the resources and services to that job. A sample SLA specification document maintained by the JobController Agent on the basis of the JRL (TSLA) and the ResourceSpecificationMemo (RSLA) is shown in Figure 5.8. Along with the other necessary information, SLA also contains client's expected completion time (ect) of a job.

```
<JRL>
<Client_Info>
       <Client_ID>cse_dept</Client_ID>
</Client_Info>
<Job_Info>
      <service_type>
             <metric>
                    <name>QoS_class</name>
                    <value>Guaranteed_Service</value>
             </metric>
       </service_type>
       <job_attributes>
             <metric>
                    <name>job_id</name>
                    <value>Job_1</value>
                    <unit></unit>
             </metric>
             <metric>
                    <name>job_type</name>
                    <value>computational</value>
                    <unit></unit>
             </metric>
             <metric>
                    <name>start_time</name>
                    <value>22-04-2007:10:00:00</value>
                    <unit>IST</unit>
             </metric>
             <metric>
                    <name>end_time</name>
                    <value>24-04-2007:10:00:00</value>
                    <unit>IST</unit>
             </metric>
             <metric>
                    <name>ect</name>
                    <value>40</value>
                    <unit>hours</unit>
             </metric>
       </job_attributes>
       <resource_attributes>
             <metric>
                    <name>os</name>
                    <value>Linux</value>
                    <unit></unit>
                    <op>EQ</op>
             </metric>
             <metric>
                    <name>num_of_processor</name>
                    <value>4</value>
                    <unit></unit>
                    <op>GE</op>
             </metric>
       </resource_attributes>
</Job_Info>
</JRL>
```

Figure 5.6: JRL.

```
<ResourceSpecificationMemo>
       <RP_Info>
              <metric>
                     <name>RP_ID</name>
                     <value>jugrid</value>
              </metric>
              <metric>
                     <name>RP_Type</name>
                     <value>computational</value>
              </metric>
              <metric>
                     <name>RP_AvailDay</name>
                     <value>2-3-4-5-6</value>
              </metric>
              <metric>
                     <name>RP_AvailTime_Day</name>
                     <value>24</value>
                     <unit>hours</unit>
              </metric>
              <metric>
                     <name>RP_NotAvailTime_Day</name>
                     <value>Nil</value>
              </metric>
              <metric>
                     <name>RP_Cost</name>
                     <value>XXX</value>
                     <unit>Rs</unit>
              </metric>
       </RP_Info>
     <Resource_Info>
              <metric>
                     <name>num_processor</name>
                     <value>8</value>
                     <unit></unit>
              </metric>
            <metric>
                     <name>mem_avail</name>
                     <value>512</value>
                     <unit>MB</unit>
             </metric>
            <metric>
                     <name>os</name>
                     <value>Linux</value>
             </metric>
            <metric>
                     <name>machine_type</name>
                     <value>x86</value>
              </metric>
       </Resource_Info>
</ResourceSpecificationMemo>
```

Figure 5.7: ResourceSpecificationMemo.

```
<SLA>
<Party>
   <metric>
       <name>Client_ID</name>
       <value>cse_dept</value>
   </metric>
   <metric>
       <name>RP_ID</name>
       <value>jugrid</value>
   </metric>
   <metric>
       <name>Resource_Broker</name>
       <value>ABC</value>
   </metric>
</Party>
<service_type>
   <metric>
    <name>QoS_class</name>
   <value>Guaranteed_Service</value>
   </metric>
</service_type>
<cost>
   <metric>
       <name>RP_cost</name>
       <value>XXX</value>
       <unit>Rs</unit>
   </metric>
</cost>
<job_attributes>
   <metric>
       <name>job_id</name>
       <value>Job_1</value>
       <unit></unit>
   </metric>
   <metric>
       <name>job_type</name>
       <value>computational</value>
       <unit></unit>
   </metric>
   <metric>
       <name>start_time</name>
<value>22-04-2007:10:00:00</value>
       <unit>IST</unit>
   </metric>
   <metric>
       <name>end_time</name>
<value>24-04-2007:10:00:00</value>
       <unit>IST</unit>
   </metric>
   <metric>
       <name>ect</name>
       <value>40</value>
       <unit>hours</unit>
   <metric>
</job_attributes>
<resource_attributes>
   <metric>
       <name>os</name>
       <value>Linux</value>
       <unit></unit>
       <operation>EQ</operation>
   </metric>
   <metric>
       <name>num_of_processor</name>
       <value>4</value>
       <unit></unit>
       <operation>GE</operation>
   </metric>
</resource_attributes>
</SLA>
```

Figure 5.8: SLA between client, resource provider and resource broker.

5.7 Resource Brokering in PRAGMA

The three main stages of resource brokering, namely Job Modeling, Resource Discovery and Resource Selection have already been mentioned in Section 5.1, and the design aspects of these three stages in the context of the multi-agent system have been elaborated in Sections 5.2, 5.3, 5.4 and 5.5. In addition, Section 5.6 introduces service level agreements established within the MAS in order to maintain the quality of service. This section highlights the implementation details of the resource broker component in PRAGMA, which has been introduced in Section 4.3.

Job Modeling

At the first stage of resource brokering, it is necessary to describe the resource requirement of the job at the time of execution. As shown in Figure 5.9, an object of `PRAGMA_JobRequirement` class prepares a JRL from the description of each job (JDesc) of a batch and identifies the minimal requirement to run that job. It also maintains the clients' information that is later used to establish the SLA between client, resource provider and resource broker.

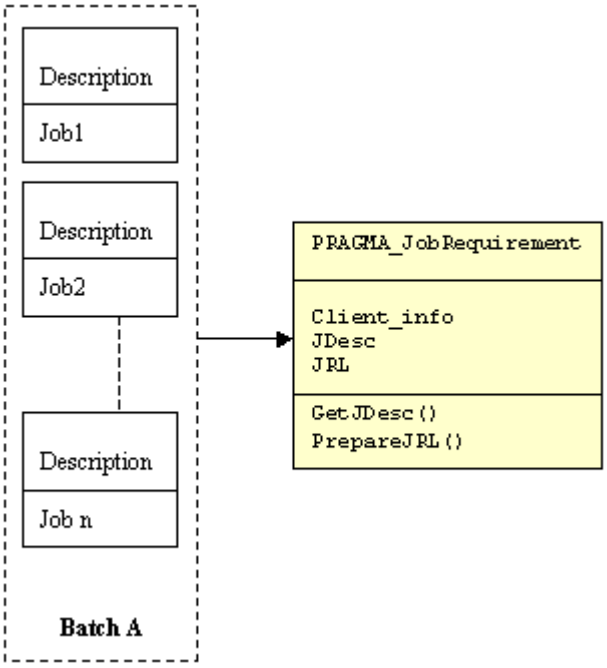


Figure 5.9: Class diagram of job modeling performed by `PRAGMA`.

Gathering Resource Information

In order to select suitable resource providers, which can fulfill the requirements of a job, ResourceProvider Agent of `PRAGMA` needs to collect resource information with the help of some Grid information provider. Thus, information providers, like in other distributed environments, play an important role in resource brokering in Grid. The resource broker component of `PRAGMA` takes decisions on the basis of the resource information (that it gathers from the information providers) and the client's requirements (as specified in the

JRL). PRAGMA employs the MDS component of Globus Toolkit 4 and Ganglia information provider for this purpose.

Globus toolkit offers information services through its Monitoring and Discovery Service component (MDS). MDS defines and implements mechanisms for service and resource discovery and monitoring on Grid. MDS provides information about the accessible resources on the Grid. The recently released version MDS4, as a part of the Globus Toolkit 4 [41], implements a standard web service interface. MDS4 builds on query, subscription and notification protocols and interfaces defined by the WS Resource Framework (WSRF) and WS Notification. MDS4 includes two higher-level services, *Index Service* and *Trigger Service.* Index Service collects and publishes aggregated information about Grid resources and Trigger Service collects resource information and performs actions when certain conditions are triggered. These services are built upon a common infrastructure called the *Aggregation Framework* that provides common interfaces and mechanisms for working with data sources. A web based user interface to the Index Service is provided by WebMDS. MDS4 aggregator source obtains data from an *information provider* that is an external component and publishes into its aggregator service [47]. Globus Toolkit 4 (GT4.0) comes with Monitoring and Discovery Service (MDS4), which includes Ganglia [38] and Hawkeye [50] as external information providers. These are configured in the XML file stored as $GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp/gluerp.xml [47].
In order to collect data from Ganglia, the default provider option in this file is set as:

```
<defaultProvider>
java org.globus.mds.usefulrp.glue.GangliaElementProducer
</defaultProvider>
```

Ganglia not only provides information about the host on which it is running, but also it collects information about all the hosts participating and offering computational services in the Grid. Ganglia can also be used to monitor workload of Grid resources. It registers the information with Globus MDS. Based on this workload information, suitable resources are

chosen for executing the applications. With the help of Globus MDS (Monitor and Discovery Service) [41] and Ganglia information provider [38], the information about resources and their quality is collected by PRAGMA. The services offered by Ganglia are discussed in detail in the following paragraphs.

Ganglia Information Provider

Ganglia is an open source monitoring and execution environment developed at the University of California, Berkeley Computer Science Division. It is a scalable monitoring system for high-performance, distributed computing environments such as clusters and Grids [38, 67]. Ganglia comprises two daemons, namely *Ganglia Monitoring Daemon (gmond)* and *Ganglia Meta Daemon (gmetad).* Gmond, which is used by the ResourceProvider Agent in PRAGMA, is a multi-threaded daemon that runs on each Grid node. Gmond monitors each Grid node and listens to the state of all other nodes through a unicast or multicast channel. Gmond transmits information using two different techniques. The first one is unicasting/multicasting host state in XDR (eXternal Data Representation) format using UDP messages and the second is sending XML messages over a TCP connection. PRAGMA implementation uses the second technique in order to gather the resource information. The Ganglia PHP-based frontend provides a view of the gathered information via real-time dynamic web pages [38]. Detailed discussion on the gmetad, Ganglia PHP web frontend and other utility programs are not within the scope of this thesis. The resource information is collected as metric value pairs. Some of the useful metric names and their descriptions are shown in Table 5.1.

In the remaining part of this section, the resource discovery and resource selection performed by PRAGMA are described in details.

Resource Discovery and Selection

Figure 5.10 shows how ResourceProvider Agent collects XML data (containing resource information) over TCP from different nodes in Grid. In case of a virtual organization, once the client submits the job at any node of a Grid within a concrete organization (Figure 2.1),

the ResourceProvider Agent collects resource information of that Grid as well as of other Grids in other organizations with the help of other ResourceProvider Agents as shown in Figure 5.11. The `PRAGMA_ResourceProviderAgent` class is shown in Figure 5.12, which is used to collect the resource related information with the help of Ganglia information provider.

Metric Name	**Description**
Host_name	Name of the host
IP	IP of the host
Boottime	System boot timestamp
Sys_clock	Current time on host
cpu_idle	Percent CPU idle
cpu_speed	Speed in MHz of CPU
cpu_num	Number of CPUs
Mem_buffers	Amount of buffered memory
Mem_shared	Amount of shared memory
Mem_total	Amount of available memory
Os_name	Operating system name
disk_free	Total free disk space
disk_total	Total available disk space

Table 5.1: Resource information.

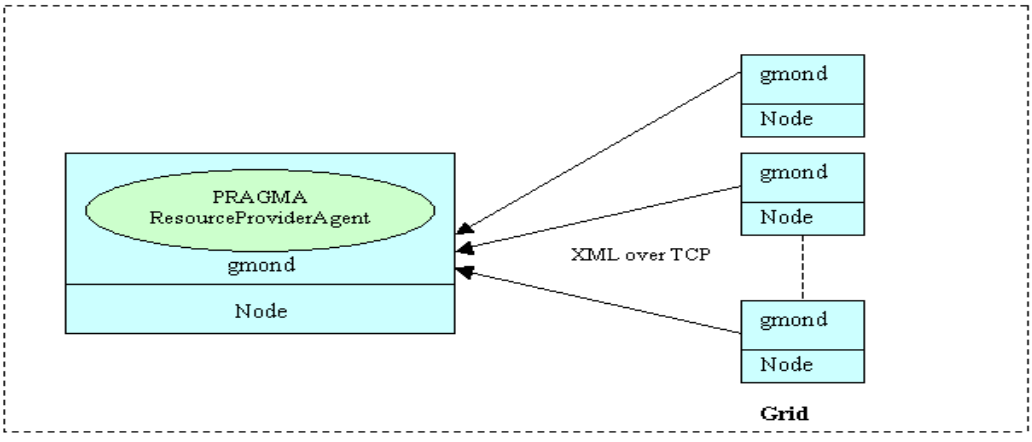


Figure 5.10: Collecting resource information using Ganglia gmond daemon in Grid.

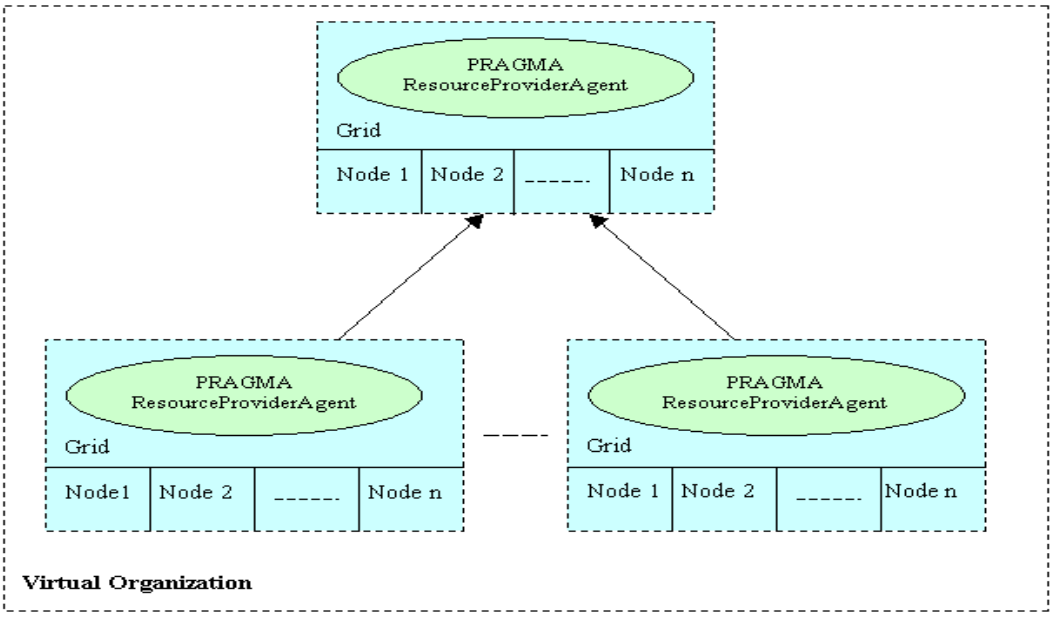


Figure 5.11: Collecting resource information in VO.

The `PRAGMA_JobControllerAgent` class of `PRAGMA` selects an appropriate resource provider for each job of a batch with the objective of overall cost optimization. This class is also responsible for establishing *Service Level Agreements (SLA)* between the client and the resource owners with the help of JRL and ResourceSpecificationMemo as discussed in the previous sections. It decides the optimal mapping of all the concurrent jobs onto the Grid sites and accordingly creates a JobMap. Once a resource provider is selected for a job, the `PRAGMA_JobControllerAgent(JCA)` initiates a `PRAGMA_JobExecutionManagerAgent(JEM)`, which is a mobile agent. It starts the execution of the job on a remote resource provider as specified in the JobMap. `PRAGMA_JobControllerAgent` also maintains the JobMap and keeps track of the SLAs for each of the concurrent jobs submitted by a client. Agent class diagram of `JCA` and `JEM` is shown in Figure 5.13. Figure 5.14 and 5.15 depict the sequence diagrams related to the three phases of resource brokering. However, rescheduling scenario is not included in these diagrams.

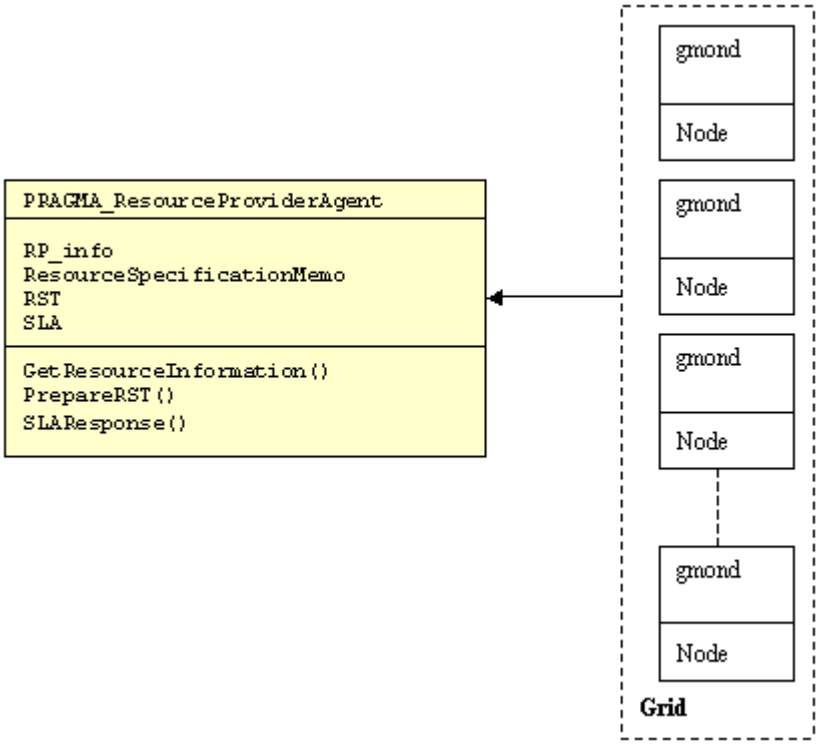


Figure 5.12: Agent class diagram for resource discovery.

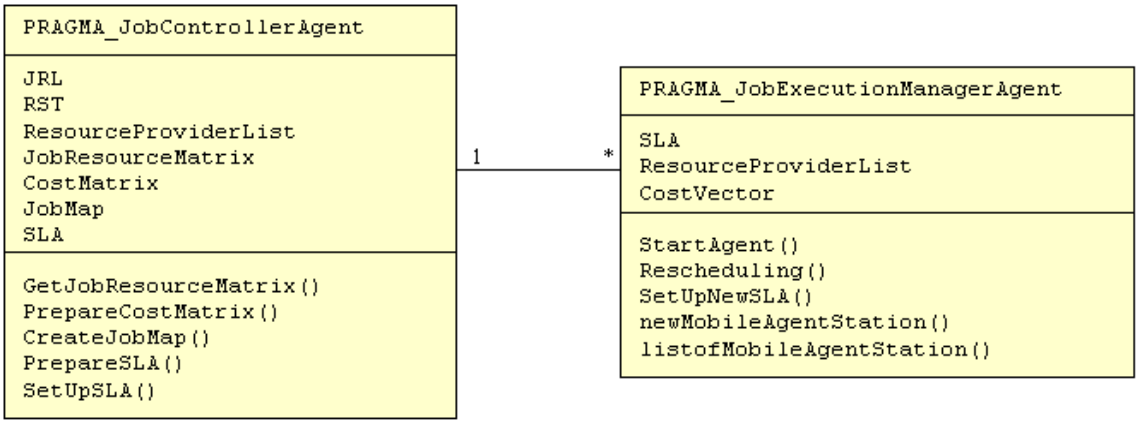


Figure 5.13: Agent class diagram of JCA and JEM.

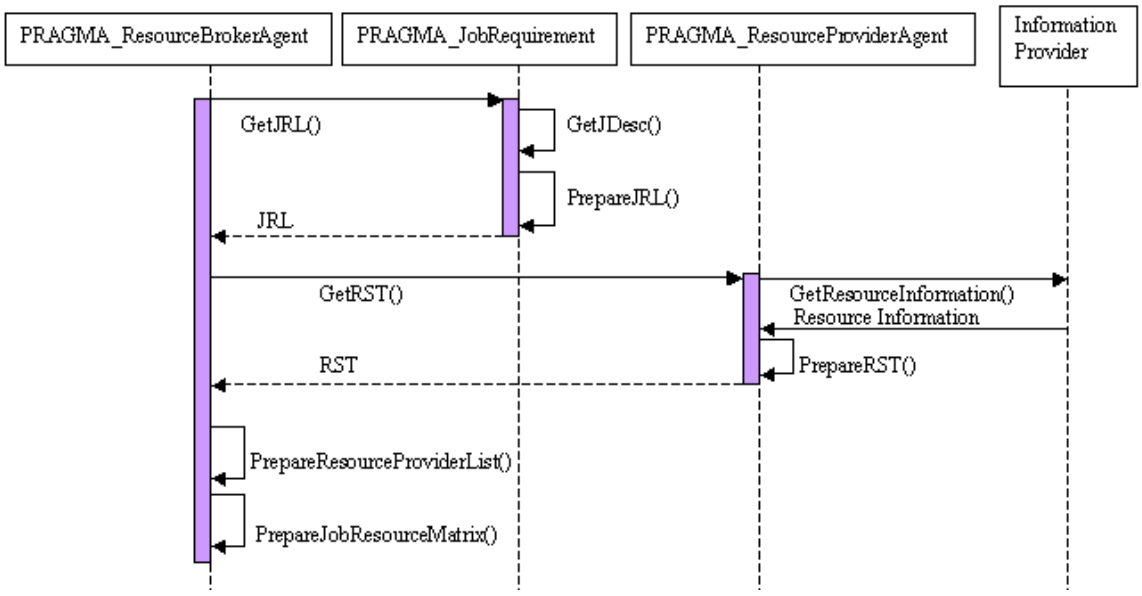


Figure 5.14: Sequence diagram of job modeling and resource discovery.

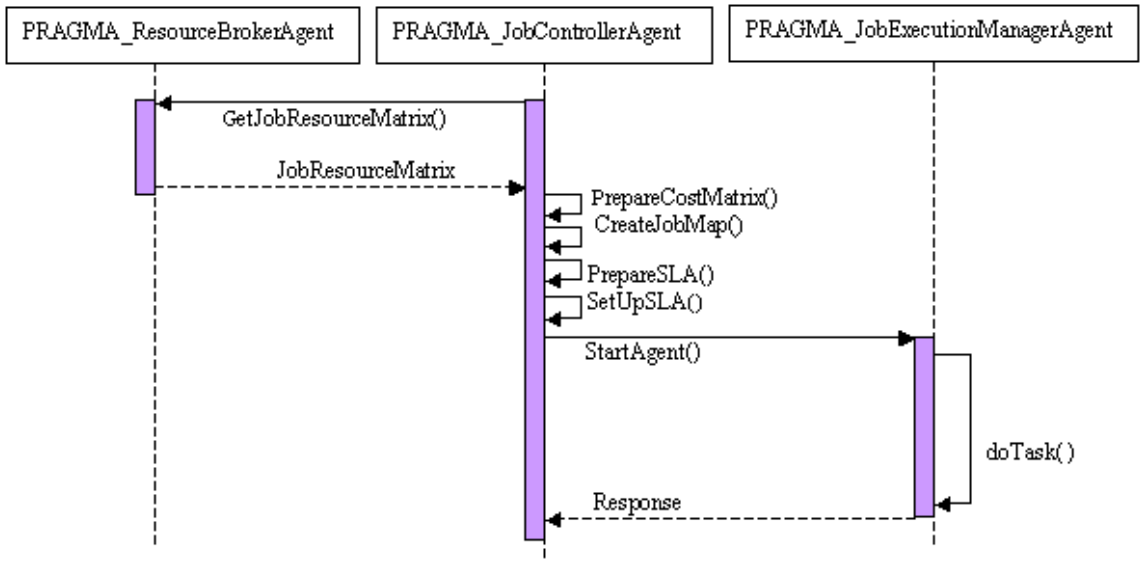


Figure 5.15: Sequence diagram of resource selection.

At the time of rescheduling, the `PRAGMA_JobExecutionManagerAgent` controls and keeps track of the execution of a job and does the necessary resource brokering activities. Thus a `PRAGMA_JobExecutionManagerAgent` liaises with the

PRAGMA_AnalysisAgent and takes necessary actions as soon as it receives a WarningMessage from the PRAGMA_AnalysisAgent regarding an SLA violation (or overprovisioning) for which rescheduling is necessary. With the help of ResourceProviderList and CostVector, PRAGMA_JobExecutionManagerAgent selects a new resource provider (see Section 5.5) and obtains an SLA and reschedules the job there. Chapter 8 addresses the issues and implementation details related to rescheduling and local tuning of the jobs.

5.8 Conclusion

This chapter discusses the issues related to *resource brokering* of multiple concurrent jobs within a multi-agent system. Resource brokering plays an essential role to optimize the resource usage in a Grid environment. The importance and requirements of a resource broker are highlighted in this chapter. Main focus of this chapter is on the approaches at different stages of resource brokering and strategies in order to optimize the overall cost for resource usage while maintaining the required QoS. The chapter presents a novel algorithm to select resource providers for a batch of jobs with the goal of overall cost optimization. This approach is essentially better that the approaches taken by other resource brokers for minimizing the cost of a particular job at any particular time. In order to maintain the QoS, MAS supports rescheduling of a job running on a particular host to a different host in case of performance degradation. Resource brokering at this stage has also been discussed in this chapter. Implementation details of the resource brokering component of PRAGMA (a tool, which implements the proposed MAS) are also discussed in this chapter.

The next chapter presents the experimental framework that is used for demonstrating the efficiency of the resource brokering component in PRAGMA and also presents the evaluation results of resource brokering strategies which have been described in this chapter.

Chapter 6

Evaluation of Resource Brokering in PRAGMA

Chapter 4 introduces a tool PRAGMA (**P**erformance based **R**esource Brokering and **A**daptive execution in **G**rid for **M**ultiple concurrent **A**pplications), which has been developed to demonstrate the efficiency of the multi-agent system proposed in this thesis. A significant characteristic of PRAGMA in comparison with other existing resource management systems is that it provides support for *multiple concurrent executions* of a batch of jobs in Grid. This characteristic is featured in Chapter 5 at the time of designing and implementing the resource brokering services of PRAGMA. New strategies and algorithms for resource brokering have been incorporated and implemented in PRAGMA that have been described in Chapter 5.

This chapter evaluates the algorithms described in the previous chapter and demonstrates the effectiveness of PRAGMA. A local Grid test bed has been setup for this purpose. The test bed consists of heterogeneous nodes. Currently all these nodes are running Linux. However, as Java has been used for implementing the agent-based system, interoperability of the system can be assured. Major computational nodes of the setup include IBM Power4 pSeries 690 (P-690) Regatta server and HP NetServer LH 6000. Other nodes are Intel Core2 Duo PCs and Intel Pentium-4 PCs. GT4 is installed on all nodes and Ganglia information provider is associated with each of them. Communication takes place with the help of a fast local area network. Experiments have been carried out on this local Grid test

bed with a goal of assessing the performance of resource brokering algorithms in PRAGMA at the time of resource selection and allocation for multiple concurrent jobs.

Section 6.1 demonstrates a Grid portal for carrying out the experiments for resource brokering in PRAGMA. The experimental results obtained in different phases of resource brokering by PRAGMA, including job modeling, resource discovery, resource selection, and SLA establishment are presented in the following sections.

6.1 PRAGMA Grid Portal

Complex strategies are implemented within PRAGMA to accomplish resource brokering with the help of agents (see Chapter 4 and Chapter 5). A Grid portal hides all these complexities of PRAGMA from a client and provides an interface to submit jobs that will use the resources and services provided by the Grid. It also hides the underlying agent interactions and their working procedures. From this perspective, the client sees the Grid as a virtual computing resource. A Grid-enabled web portal gives clients a familiar and consistent way to interface with complex underlying services using a standard web browser. It allows scientists or engineers to focus on their problem area by making the Grid a transparent extension of their desktop computing environment. Graphical user interface shows information in a more sophisticated way. Workflow management is also simple and straightforward. Figure 6.1 shows the high-level view of the PRAGMA Grid portal application flow.

Grid portal provides a single access point through which authentication and authorization can be checked. Clients can sign in to the Grid through the portal, using their credentials, which are verified by the portal to check whether the clients are valid Grid users. PRAGMA Grid portal has a login screen as shown in Figure 6.2. PRAGMA Grid portal allows the client to access the Grid only if it has an entry in the gridmap file (see Chapter 3). This process is used to authenticate users of PRAGMA Grid portal and it is not a part of GSI mechanism (enhanced security features may be included at a later stage). Figure 6.3 demonstrates the

PRAGMA Grid portal login flow. After the client has successfully authenticated with a user ID and password, the Grid job submission screen is presented, as shown in Figure 6.4.

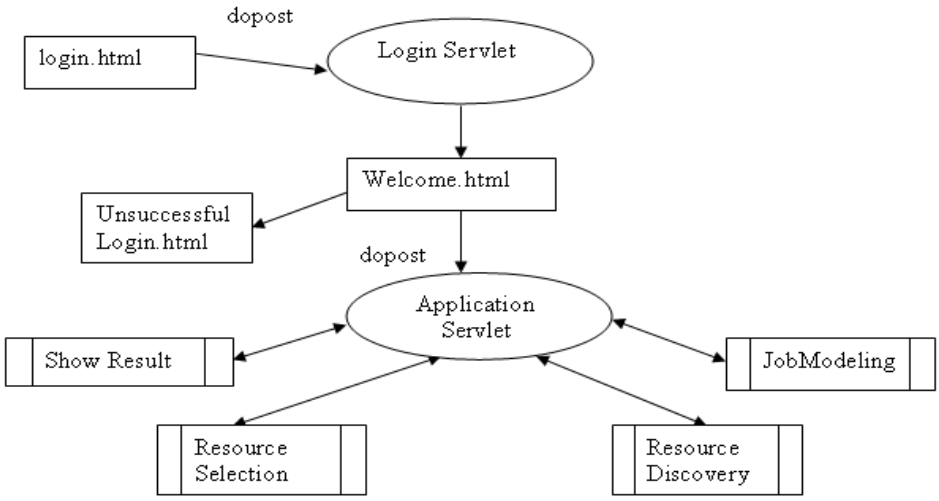


Figure 6.1: PRAGMA Grid portal application flow.

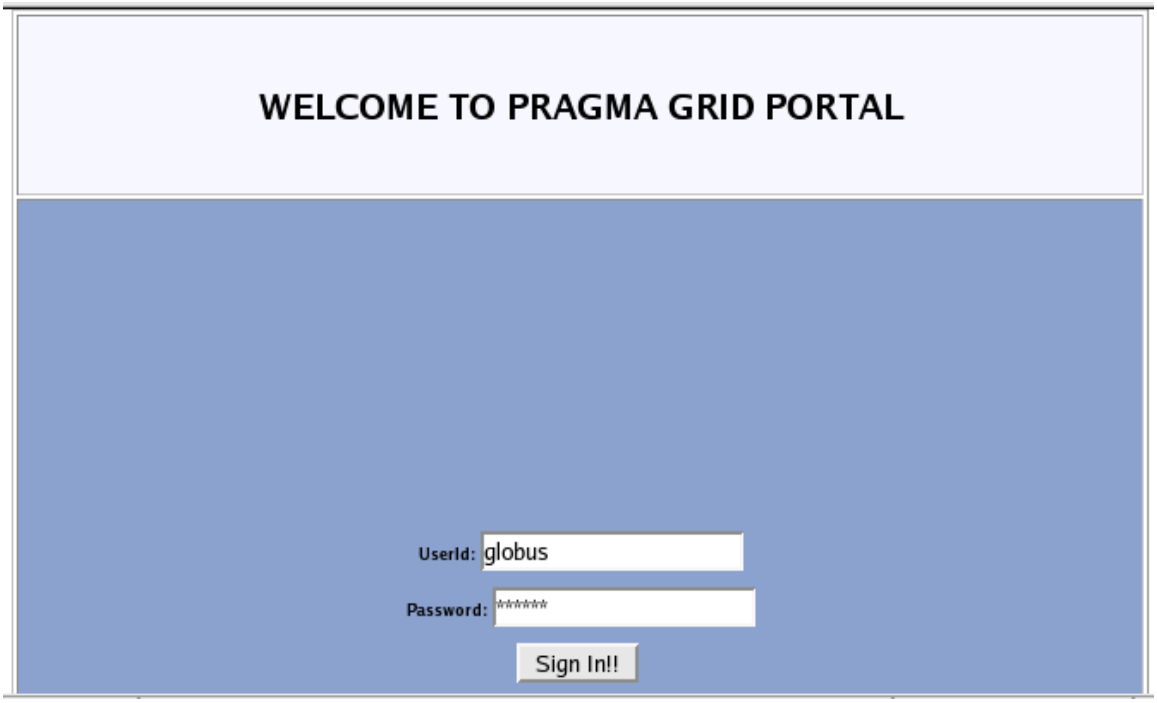


Figure 6.2: PRAGMA Grid portal login screen.

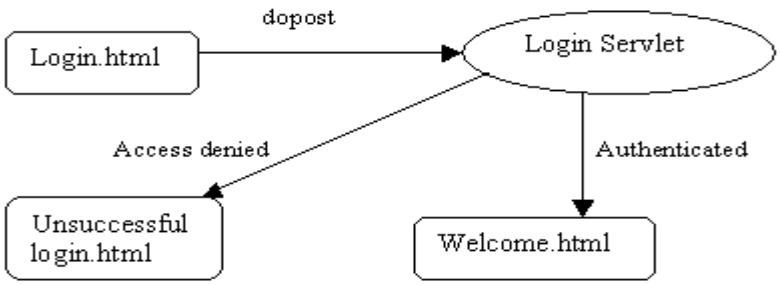


Figure 6.3 PRAGMA Grid portal login flow.

PRAGMA mainly considers four types of jobs - sequential Java applications, sequential C applications, parallel Java applications (Jomp) and parallel C applications (OpenMP). The client can submit a job by selecting the job type (Java, C/C++, Jomp, C with OpenMP directives) from the job submission screen. Different phases of resource brokering are carried out and suitable resource providers are allocated to each job in the batch. The phases include job modeling, resource discovery, resource selection, and finally SLA establishment.

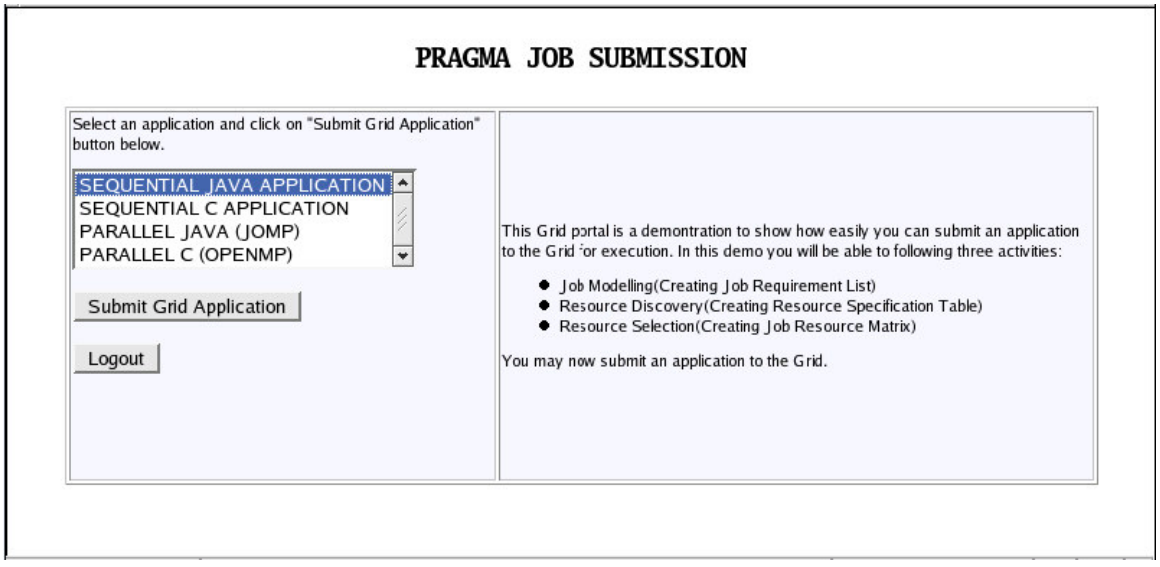


Figure 6.4: PRAGMA Grid portal job submission screen.

6.2 Job Modeling by PRAGMA

Details of job modeling and the structure of JRL have been discussed in Chapter 5. In the job modeling phase, the requirements of a job are stored in XML format representing the JRL (see Section 5.2 and 5.7) of that particular job. For example, JRL of the i^{th} job is stored in JRL_i.xml file. Figure 6.5 shows an example XML file of JRL. This XML file for JRL specifically contains client information and job's resource requirement information. For example, circled area in Figure 6.5 shows client information like Client_ID, job description which includes service type, job id, job type, start time, end time, ect (expected completion time), instruction count, memory reference, miss rate and job requirements as specified by the client like machine_type, OS, min_free_mem, num_of_processor, cpu_speed etc. Expected completion time of this example job is 40 hours according to client's perception. JRL file for each job is later used by PRAGMA to select Resource Providers, which can fulfill the requirements of the job as well as used as TSLA to establish SLA between the client and the selected resource provider.

```
<?xml version= "1.0"?>
<JRL>
<Client_Info>
        <Client_ID>cse_dept</Client_ID>
</Client_Info>
<Job_Info>
        <service_type>
                <metric>
                        <name>QoS_class</name>
                        <value>Guaranteed_Service</value>
                </metric>
        </service_type>

<job_attributes>
        <metric>
                <name>job_id</name>
                <value>Job_1</value>
                <unit></unit>
        </metric>
        <metric>
                <name>job_type</name>
                <value>computational</value>
                <unit></unit>
        </metric>
        <metric>
                <name>start_time</name>
                <value>22-04-2007:10:00:00</value>
                <unit>IST</unit>
        </metric>
        <metric>
                <name>end_time</name>
                <value>24-04-2007:10:00:00</value>
                <unit>IST</unit>
        </metric>
        <metric>
                <name>ect</name>
                <value>40</value>
                <unit>hours</unit>
        </metric>
        <metric>
                <name>instruction_count</name>
                <value>53108947985</value>
                <unit></unit>
        </metric>
        metric>
                <name>mem_ref</name>
                <value>1166</value>
                <unit></unit>
        </metric>
        metric>
                <name>miss_rate</name>
                <value>60.38</value>
                <unit>%</unit>
        </metric>
</job_attributes>

        <resource_attributes>
                <metric>
                        <name>machine_type</name>
                        <value>x86</value>
                        <unit></unit>
                        <op>EQ</op>
                </metric>
                <metric>
                        <name>os</name>
                        <value>linux</value>
                        <unit></unit>
                        <op>EQ</op>
                </metric>
                <metric>
                        <name>min_free_mem</name>
                        <value>1</value>
                        <unit>GB</unit>
                        <op>GE</op>
                </metric>
                <metric>
                        <name>num_of_processor</name>
                        <value>4</value>
                        <unit></unit>
                        <op>GE</op>
                </metric>
                <metric>
                        <name>cpu_speed</name>
                        <value>2000</value>
                        <unit>MHz</unit>
                        <op>GE</op>
                </metric>
        </resource_attributes>
</Job_Info>
</JRL>
```

Figure 6.5: An example XML file for JRL.

6.3 Resource Discovery by PRAGMA

The next phase of resource brokering is resource discovery. Theoretical and implementation details of this phase have been discussed in Chapter 5. For the resource discovery purpose, PRAGMA needs to collect dynamic information about the available resource providers in Grid. As discussed in Section 5.7, Globus Toolkit 4 (GT4.0) supports different information providers; among them PRAGMA uses Ganglia information provider. Ganglia collects information about all the hosts participating and offering computational services in the Grid. PRAGMA prepares RST from all these gathered information (see Section 5.3 and 5.7) and store in an XML file named RST.xml. Figure 6.6 shows a part of the XML file, which contains ResourceSpecificationMemo of different resource providers.

This XML file for RST presents the status of resources from different resource providers in form of SLO attributes and SLI (as discussed in Section 2.5 and 5.6). For example, circled area in Figure 6.6 highlights the host name and IP of the resource provider along with the resource information of that resource provider which includes cpu_num, mem_total, disk_total, mem_free etc. During the resource selection phase, PRAGMA extracts the ResourceSpecificationMemo of each resource provider from RST and matches with the JRL of a particular job. ResourceSpecificationMemo of each resource provider is considered as a RSLA (as discussed in Section 5.6) and later used by PRAGMA to establish an SLA between the resource provider and the job for which the resource provider is selected.

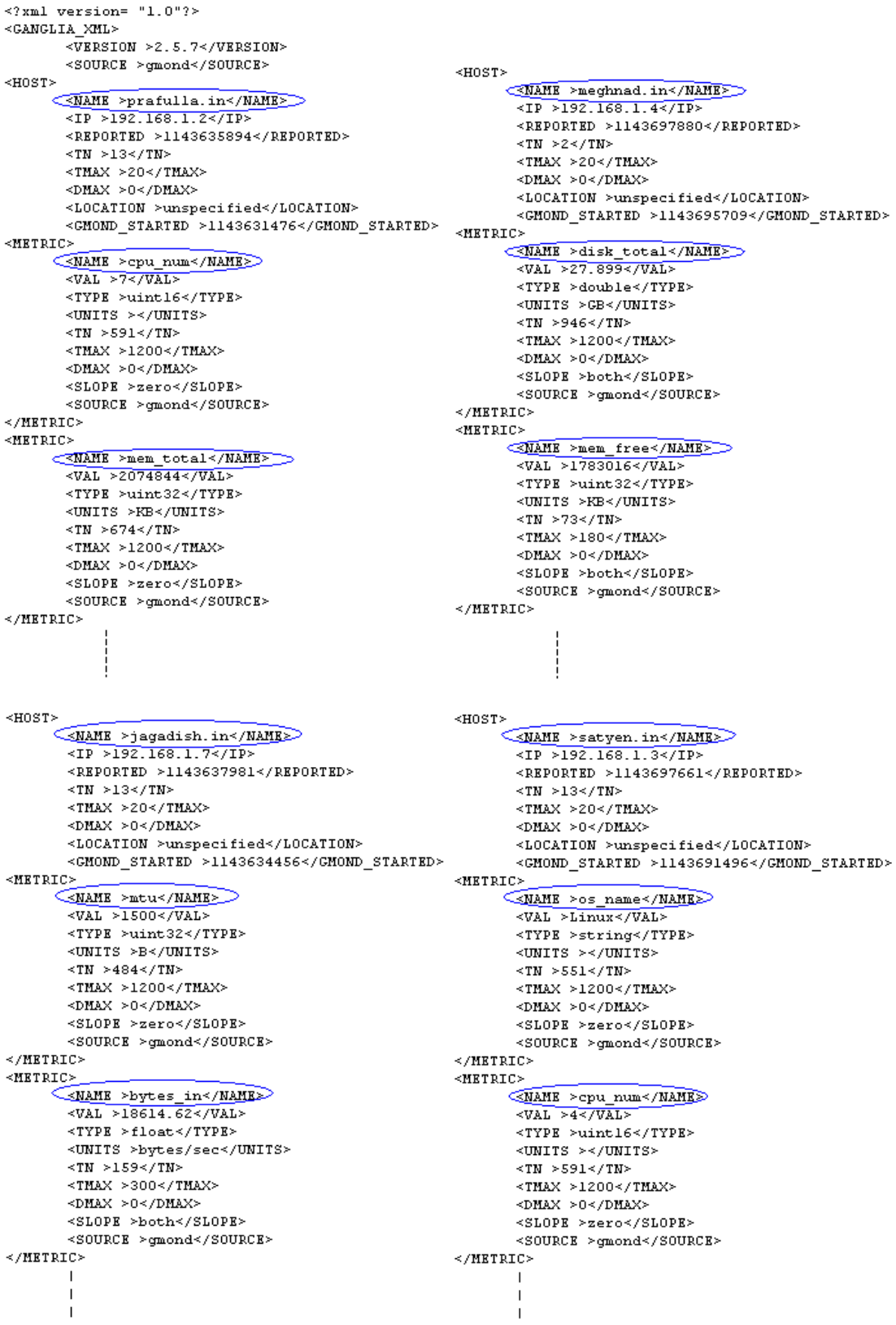


Figure 6.6: An example XML file for RST.

6.4 Resource Selection by PRAGMA

To demonstrate the performance of the strategies presented in Chapter 5, a set of experiments have been carried out on the Grid test bed as described earlier. The number of Grid resource providers has been varied from 50 to 200. The ResourceSpecificationMemo of each resource provider is obtained from Ganglia and is stored in the RST. It has been observed that on the local Grid test bed, response time from the Ganglia information provider is fairly stable for fixed number of resource providers, although, the time taken to contact Ganglia may vary considerably across different machines at widely separated locations. Thus, for a large Grid, response time of the information provider contributes a significant amount in the total time required for resource brokering. Similarly, size of the Grid also has an effect on resource discovery time and resource allocation time as the size of the RST increases with the size of the Grid. However, this experiment assumes a constant time to contact the Ganglia information provider and a constant time to create the RST as the number of resource providers is fixed. The experiment is done with different sizes of the batches of jobs. The following measurements have been taken:

IC_Time: (Information Collection Time) Time required to collect the resource information in the Grid environment, i.e., the time taken by Ganglia information provider to collect ResourceSpecificationMemos for all available resource providers in the Grid and to prepare the ResourceSpecificationTable. This experiment assumes a constant IC_Time as the number of resource providers is fixed and the experiments have been carried out on a local Grid.

RD_Time: (Resource Discovery Time) Time required to match the JRL of jobs with the RST.

RA_Time: (Resource Allocation Time) Time required to select the best set of resource providers for the batch of jobs following the algorithm presented in Section 5.4.

T_Time: Total time taken for resource brokering.

Figure 6.7, 6.8, 6.9 and 6.10 shows the effects of IC_Time, RD_Time, and RA_Time on the total time for 50, 100, 150 and 200 resource providers respectively. Number of jobs in a batch is varied from 10 to 100 as shown in the figures. It is clear from the result that the

resource allocation time (RA_Time) dominates the total time (T_Time) when the batch size increases.

Figure 6.7: Effects of IC_Time, RD_Time, and RA_Time as the number of jobs increases for 50 resource providers.

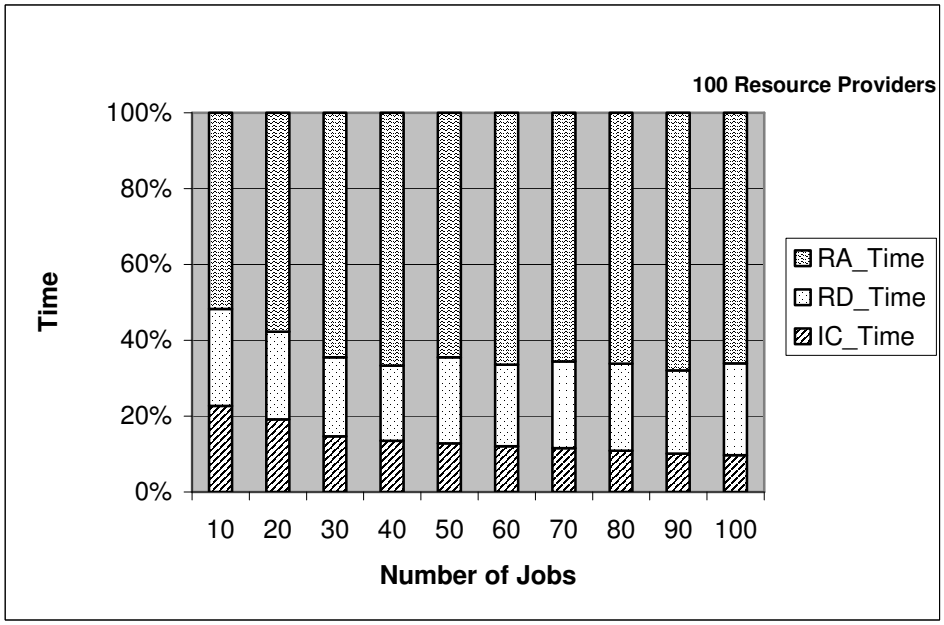


Figure 6.8: Effects of IC_Time, RD_Time, and RA_Time as the number of jobs increases for 100 resource providers.

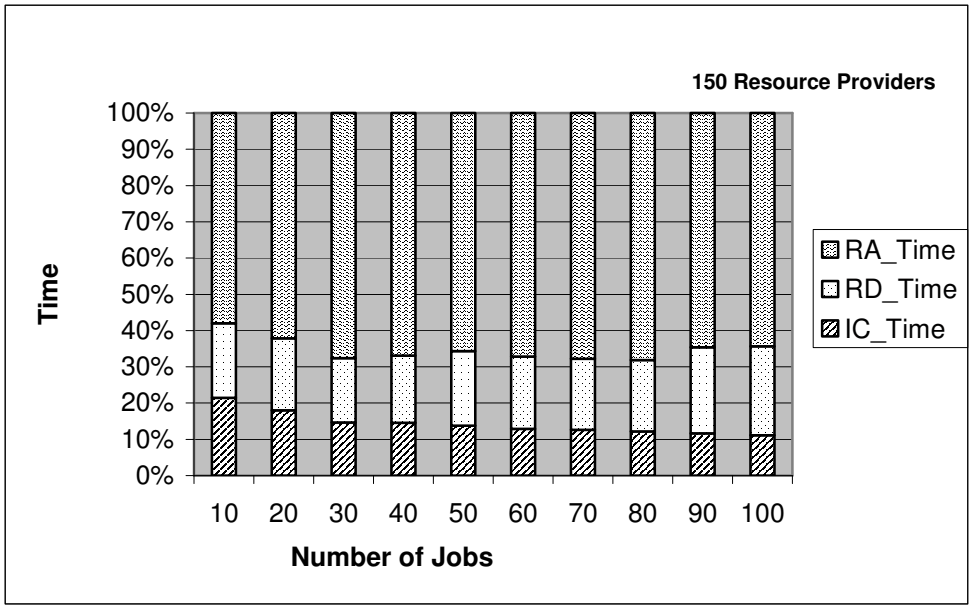


Figure 6.9: Effects of IC_Time, RD_Time, and RA_Time as the number of jobs increases for 150 resource providers.

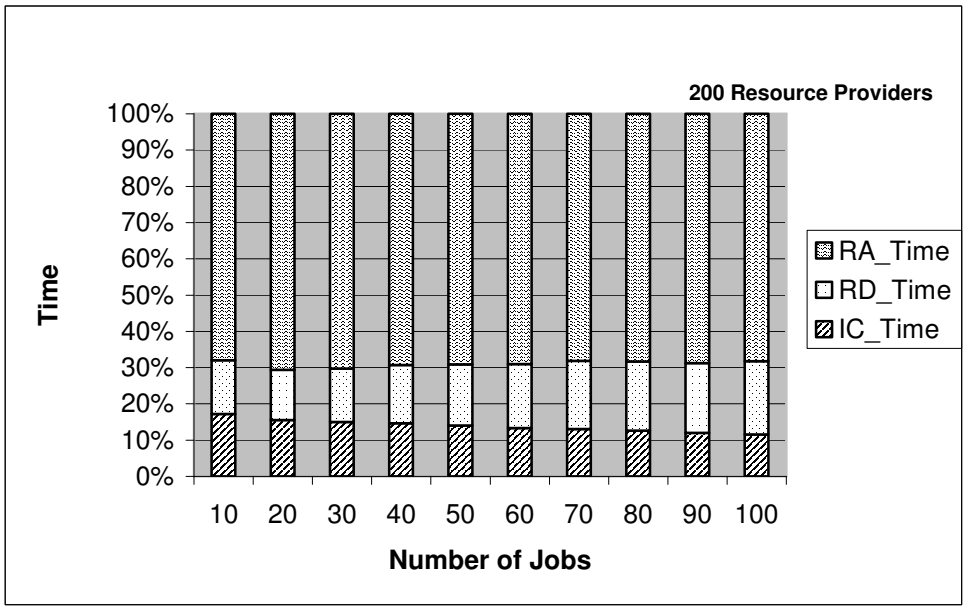


Figure 6.10: Effects of IC_Time, RD_Time, and RA_Time as the number of jobs increases for 200 resource providers.

Figure 6.11, 6.12, 6.13, and 6.14 depict the results of the same experiments, but use line diagrams, which clearly demonstrate how these times, vary with the increase in batch sizes.

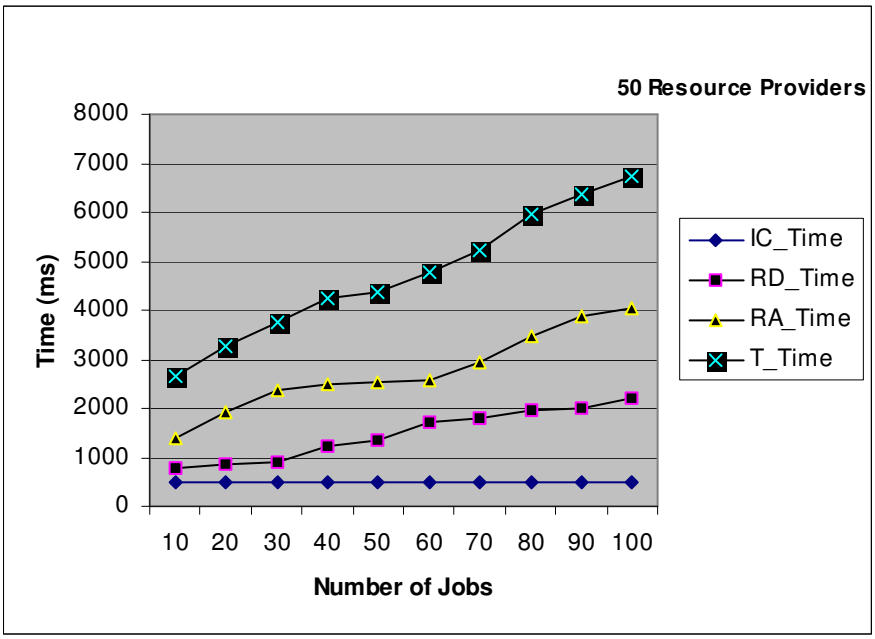


Figure 6.11: Shows the IC_Time, RD_Time, and RA_Time and T_Time as the number of jobs increases for 50 resource providers.

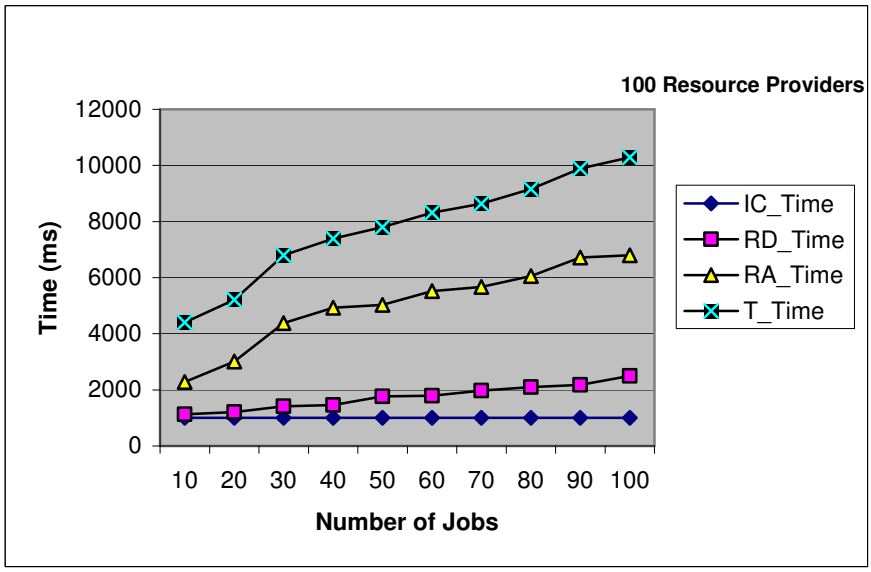


Figure 6.12: Shows the IC_Time, RD_Time, and RA_Time and T_Time as the number of jobs increases for 100 resource providers.

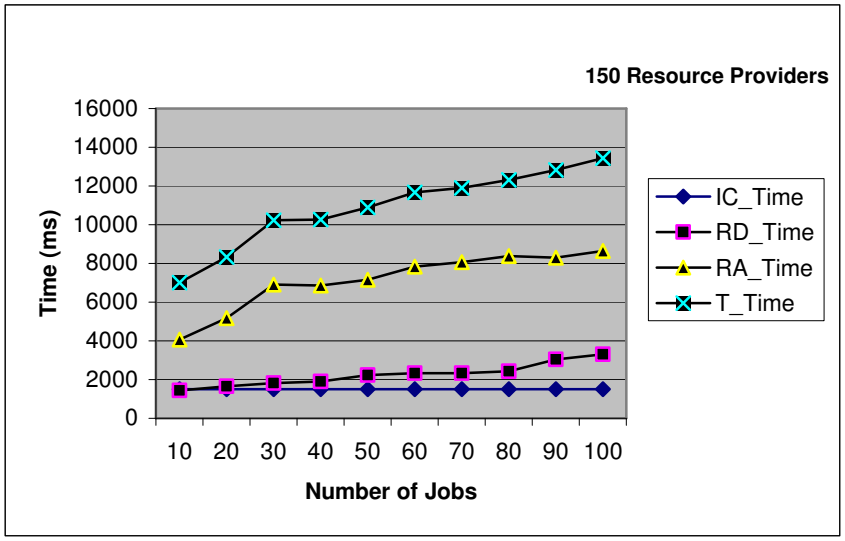


Figure 6.13: Shows the IC_Time, RD_Time, and RA_Time and T_Time as the number of jobs increases for 150 resource providers.

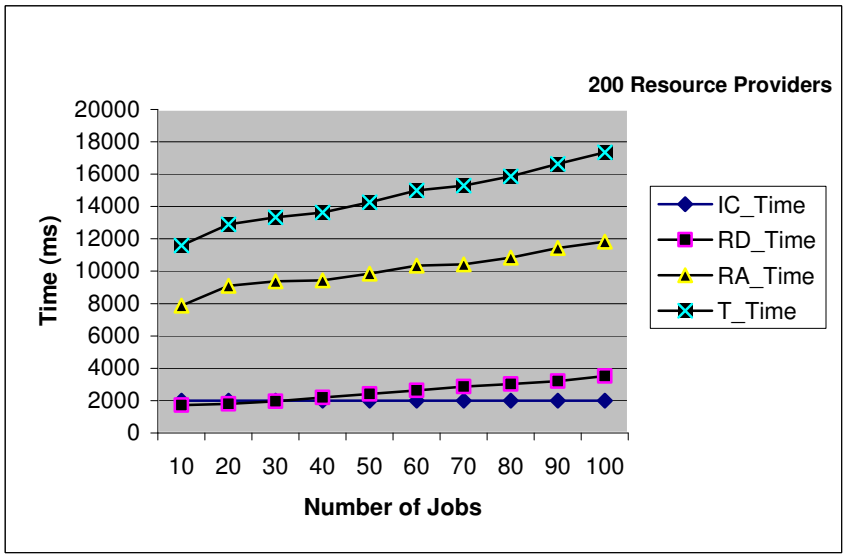


Figure 6.14: Shows the IC_Time, RD_Time, and RA_Time and T_Time as the number of jobs increases for 200 resource providers.

The algorithm presented in this thesis has also been compared with the resource selection algorithm based on *min-min* heuristic. The results of the comparison are depicted in Figure 6.15, 6.16, 6.17 and 6.18. These figures compare the estimated overall costs of executing the batches of jobs in the following two cases:

Case 1: initial resource selection algorithm is based on *Hungarian* method.

Case 2: initial resource selection algorithm is based on *min-min* heuristic.

The figures clearly show that the initial resource selection algorithm based on *min-min* heuristic is not a good choice for resource brokering. As expected, there is a large increase in the estimated resource usage cost if resource allocation for a batch of job is based on *min-min* heuristic.

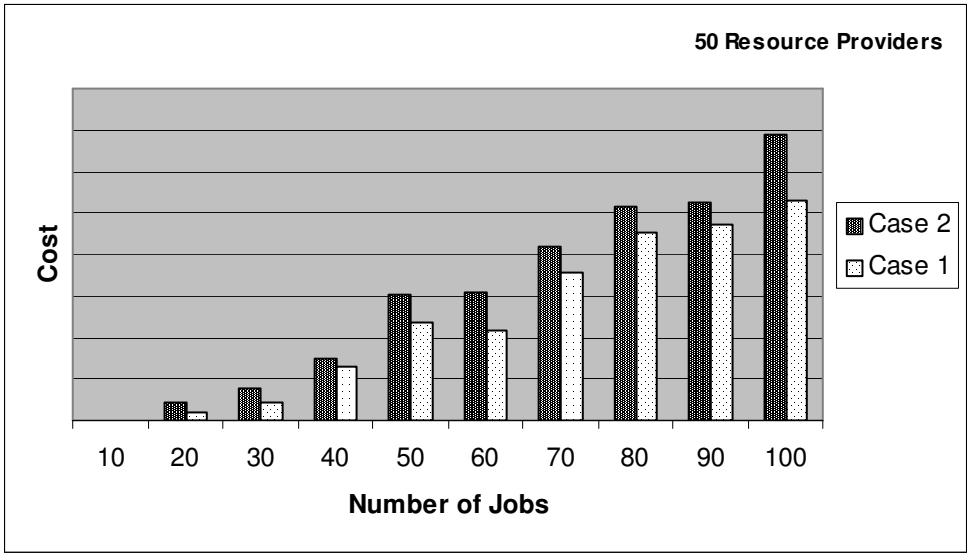


Figure 6.15: Comparison of the costs in Case 1 and Case 2 for 50 resource providers.

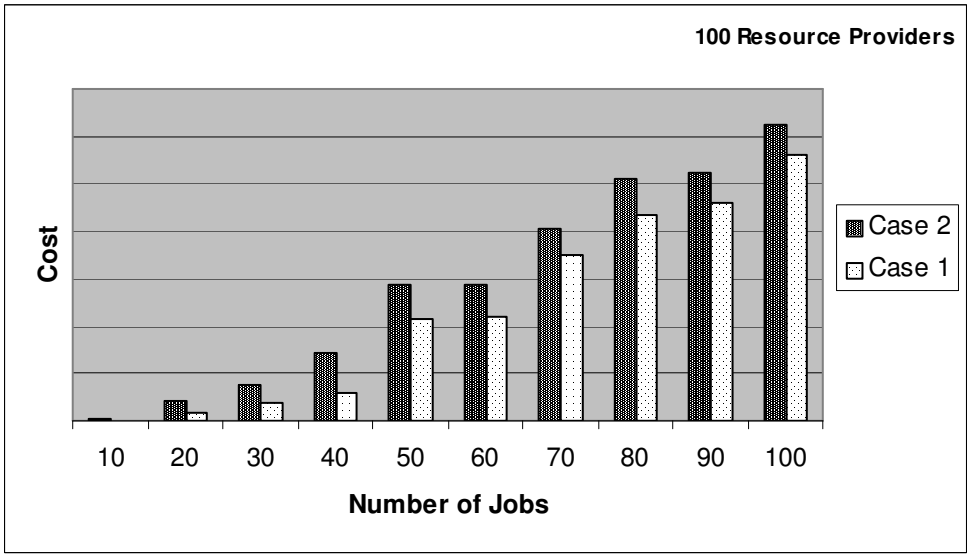


Figure 6.16: Comparison of the costs in Case 1 and Case 2 for 100 resource providers.

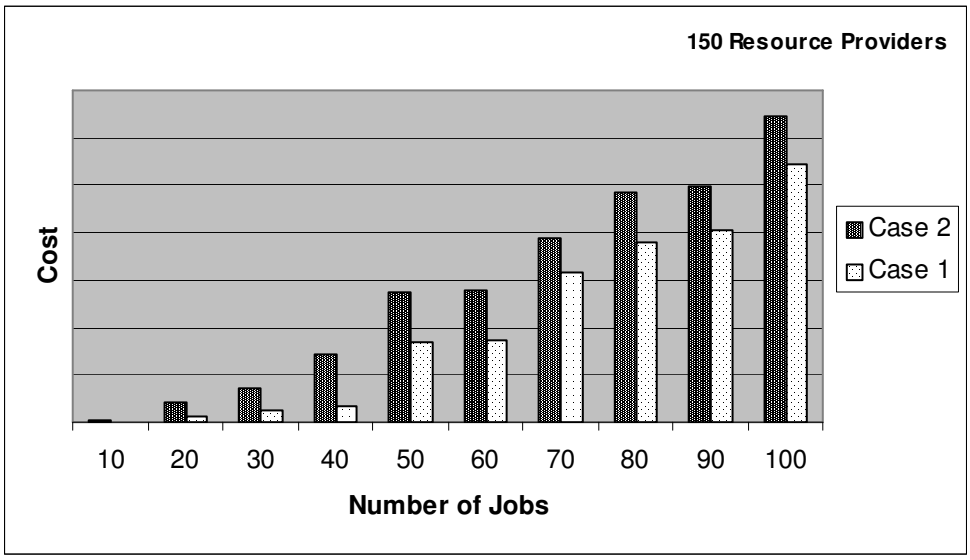


Figure 6.17: Comparison of the costs in Case 1 and Case 2 for 150 resource providers.

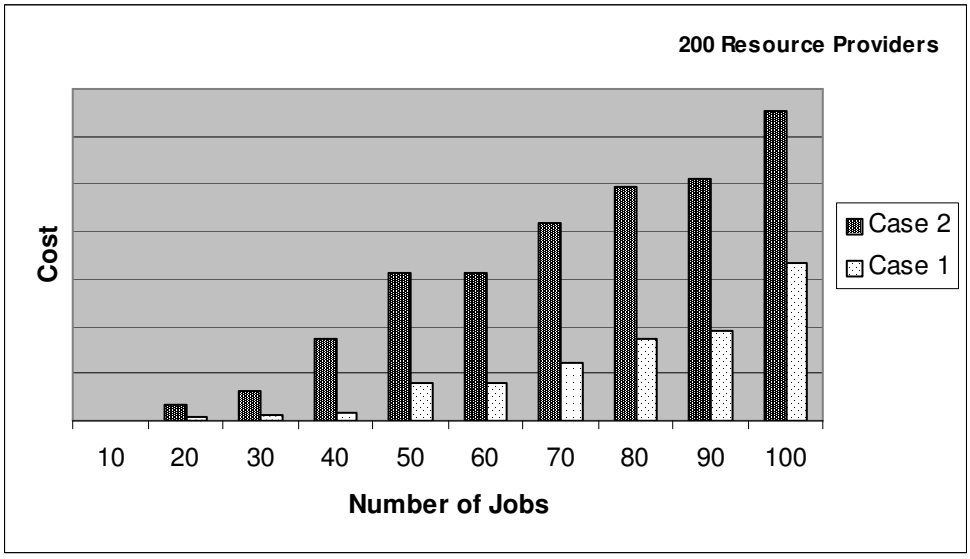


Figure 6.18: Comparison of the costs in Case 1 and Case 2 for 200 resource providers.

In particular, this set of experiments highlights that initially, while selecting resources for a batch of jobs, it is important to consider the cost effective selection of resources for all jobs in the batch, which in turn optimizes the overall resource usage cost (or time if the algorithm is modified) for the entire batch of jobs.

After completion of the resource selection procedure, PRAGMA produces a JobMap. The JobMap contains an entry for each job in the batch if there is a suitable resource provider for that particular job. It contains the Job_id and Resource_id pair for each job. Figure 6.19 shows an example of job assignment by PRAGMA using JobMap. With the help of JobMap, JRL and ResourceSpecificationMemo, PRAGMA later establishes an SLA between the client and the selected resource provider.

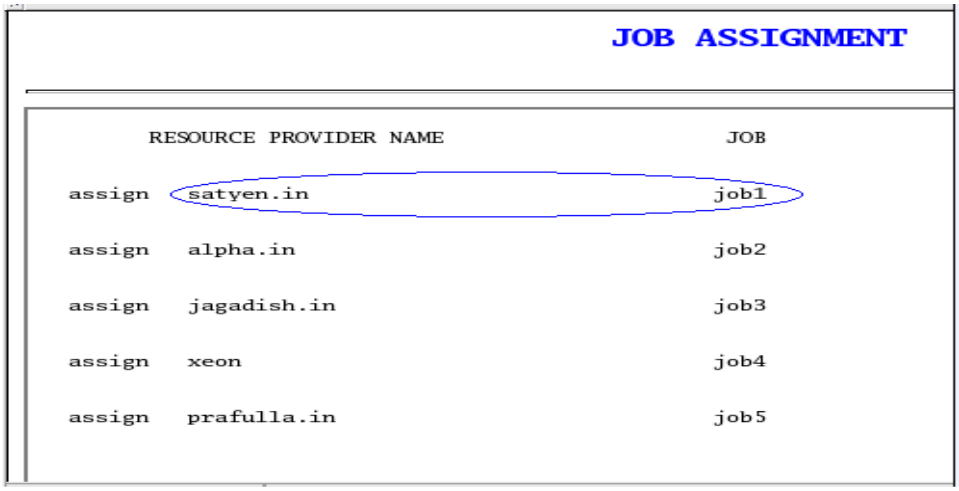


Figure 6.19: Job assignment in PRAGMA Grid portal.

6.5 SLA Establishment by PRAGMA

PRAGMA resource broker establishes an agreement with the resource provider on behalf of the client, while the resource provider guarantees that it would provide the desired quality service for that job. As discussed in Section 5.6, the JRL is considered as a TSLA and the ResourceSpecificationMemo is considered as an RSLA. Figure 6.20 shows an example SLA for a particular job (see Figure 6.5 for the JRL of this job). SLA is actually established between three parties that include the client, the resource provider and the PRAGMA resource broker as shown in the circled areas in Figure 6.20. In this example SLA, "Guaranteed-service" QoS is ensured and expected completion time is also mentioned. At the time of rescheduling, PRAGMA resource broker establishes a fresh agreement with a new resource provider.

```
<?xml version= "1.0"?>
<PRAGMA_SLA>
<Party>
        <metric>
                <name>Client_ID</name>
                <value>cse_dept</value>
        </metric>
        <metric>
                <name>RP_ID</name>
                <value>satyen.in</value>
        </metric>
        <metric>
                <name>Resource_Broker</name>
                <value>PRAGMA</value>
        </metric>
</Party>
<service_type>
        <metric>
                <name>QoS_class</name>
                <value>Guaranteed_Service</value>
        </metric>
</service_type>
<cost>
       <metric>
                <name>RP_cost</name>
                <value>XXX</value>
                <unit>Rs</unit>
        </metric>
</cost>

<job_attributes>
        <metric>
                <name>job_id</name>
                <value>Job_1</value>
                <unit></unit>
        </metric>
        <metric>
                <name>job_type</name>
                <value>computational</value>
                <unit></unit>
        </metric>
        <metric>
                <name>start_time</name>
                <value>22-04-2007:10:00:00</value>
                <unit>IST</unit>
        </metric>
        <metric>
                <name>end_time</name>
                <value>24-04-2007:10:00:00</value>
                <unit>IST</unit>
        </metric>
        <metric>
                <name>ect</name>
                <value>40</value>
                <unit>hours</unit>
        </metric>
</job_attributes>

<resource_attributes>
        <metric>
                <name>machine_type</name>
                <value>x86</value>
                <unit></unit>
                <operation>EQ</operation>
        </metric>
        <metric>
                <name>os</name>
                <value>linux</value>
                <unit></unit>
                <operation>EQ</operation>
        </metric>
        <metric>
                <name>min_free_mem</name>
                <value>1</value>
                <unit>GB</unit>
                <operation>GE</operation>
        </metric>
        <metric>
                <name>num_of_processor</name>
                <value>4</value>
                <unit></unit>
                <operation>GE</operation>
        </metric>
        <metric>
                <name>cpu_speed</name>
                <value>2000</value>
                <unit>MHz</unit>
                <operation>GE</operation>
        </metric>
</resource_attributes>
</PRAGMA_SLA>
```

Figure 6.20: SLA between client, resource provider and the PRAGMA resource broker.

6.6 Conclusion

This chapter evaluates the effectiveness of PRAGMA for resource brokering services. The results presented in this chapter indicate that the resource brokering performed by PRAGMA can generally allocate resources on the basis of optimization of overall resource usage cost for a batch of jobs. A set of experiments that compares the Hungarian method with one of the most popular algorithm based on *min-min* heuristic shows that the former gives better result than the latter for varying number of jobs, as well as varying number of resources. The experiments also measure the time required in different phases of resource brokering and demonstrate that the resource broker spends maximum time in executing the resource selection algorithm. This time also increases with the increase in number of jobs in the batch, as well as with the increase in number of resource providers. Time complexity of this algorithm is $\mathbf{O(N^3)}$, (NXN is the size of the cost matrix after converting it to a square matrix). Thus, a better algorithm should be sought for in order to optimize the initial resource allocation cost. This thesis leaves the work for future research.

To achieve the desired high performance of a job, regular performance monitoring of the job and the execution environment is important. The next chapter discusses about performance monitoring techniques and also highlights the differences in the techniques used for traditional parallel systems and for Grids. Chapter 7 also discusses the approach used by PRAGMA for performance monitoring and analysis of jobs and the environment.

Chapter 7

Performance Monitoring and Analysis

The rationale behind integrating performance monitoring services with a resource management framework has already been discussed in this thesis. However, performance monitoring of Grid infrastructure and Grid applications, by its very own nature, differs from the underlying principles of traditional performance analysis tools. The heterogeneous nature of computational resources, potentially unreliable networks connecting these resources and different administrative management domains pose the main challenge to the performance monitoring / engineering community in computer science research. Introduction of this new paradigm in the world of high performance computing has forced the computer scientists to look into the performance-engineering problem with an entirely different view. With this changing scenario, the requirements for discovering new techniques have come to the front. The requirement for efficient software support has also been felt by the researchers and much attention has been paid to developing new generation monitoring tools to meet up the requirement.

This chapter focuses on the performance monitoring aspect in Grid environment with particular emphasis on its role in QoS maintenance. The chapter first presents a survey on the performance analysis tools for traditional parallel and distributed systems (Section 7.1) and draws attention to their shortcomings with respect to the requirements of Grid performance monitoring and analysis tools (Section 7.2). In order to develop an understanding about the current research scenario, the chapter then concentrates on a set of

representative performance analysis tools for Grid environment (Section 7.3). Section 7.4 compares the features of Grid-based tools with the traditional tools and highlights the recent trends in Grid-based tools. Finally, Section 7.5 describes the implementation of performance monitoring and analysis component in PRAGMA.

7.1 Performance Monitoring in Traditional Parallel Systems

Computer scientists and researchers have a long-time target to develop effective methods and tools for performance analysis of compute-intensive applications and providing support for performance tuning. In other words, performance engineering of such applications in a cost-effective manner has been a major focus of the researchers during the last decade. Many tools for measuring or predicting performance of serial / parallel programs have been developed. These tools were designed with diverse objectives, targeted different parallel architectures and adopted various techniques for collection and analysis of performance data. The scope of these tools comprises instrumentation, measurement (monitoring), data reduction and correlation, analysis and presentation and finally, in some cases, optimisation. This section discusses the performance monitoring and analysis techniques adapted for use on traditional parallel systems and also present case studies on three performance analysis tools with an emphasis on their important features. It should be mentioned that these tools are only representative and there is no particular reason for selecting only these three tools.

7.1.1 SCALEA

SCALEA is a performance analysis tool developed at the University of Vienna [106]. It provides support for automatic instrumentation of parallel programs, measurement, analysis and visualization of their performance data. The tool has been used to analyse and to guide the application development by selectively computing a variety of important performance metrics, by detecting performance bottlenecks, and by relating performance information back to the input program.

SCALEA can be used to maintain and analyse the performance of many different programming paradigms ranging from High Performance Fortran (HPF) to data parallel, task parallel and message passing programs (OpenMP and MPI).

The components of *SCALEA* include *SCALEA Instrumentation System (SIS)*, *SCALEA Runtime System*, *SCALEA Performance Data Repository*, and *SCALEA Performance Analysis and Visualization System*. Each of these components can also be used as a separate tool. The *SCALEA Instrumentation system (SIS)* enables the user to select code regions of interest and automatically inserts monitoring code to collect all relevant performance information during an execution of the program. An execution of a program on a given target architecture is referred to as an experiment. Unlike most performance tools, *SCALEA* supports analysis of performance data obtained from individual experiments, as well as from multiple experiments. All the important information about performance experiments including source code, machine information and performance results are stored in a data repository.

SCALEA focuses on four major categories of temporal overheads. These include (a) data movement overhead that corresponds to local and remote memory accesses, file I/O and communication, (b) synchronisation overhead due to barriers and locks, (c) control of parallelism overhead (time required for fork and join or loop scheduling), and (d) additional computation caused by algorithmic or compiler changes to increase parallelism or data locality. The classification of overheads in *SCALEA* is hierarchical.

7.1.2 Paradyn

Paradyn, from the Computer Sciences Department of University of Wisconsin, Madison, is a tool for measuring and understanding the performance of serial, parallel and distributed programs [69, 87]. It consists of a data collection facility (the Data Manager), an automated bottleneck search tool (the Performance Consultant), a data visualization interface (the Visualization Manager), and a User Interface Manager. The central part of the tool is a

multi-threaded process and communication between threads is defined by a set of interfaces constructed to allow any module to request a service of any other module.

The performance consultant module of *Paradyn* automates the search for a predefined set of performance bottlenecks based on a well-defined model, called the W^3 *search model*. It attempts to find the answer of three questions: firstly why the application is performing poorly, secondly where the performance problem is located, and finally when the problem does occur. To answer the "why" question, tests are conducted to identify the type of bottleneck (e.g., synchronization, I/O, computation). By answering to the "where" question a performance bottleneck is associated with a specific part of the program or a resource used by it (e.g., a disk system, a synchronization variable, or a procedure). Answering the "when" question, the tool identifies a bottleneck with a specific phase of the program's execution.

Paradyn uses *dynamic instrumentation* to instrument only those parts of the program relevant for finding the current performance problem. Dynamic instrumentation defers instrumenting the program until it is executed and dynamically inserts, alters and deletes instrumentation during program execution. Which data to be collected is also decided during the execution of the program under the guidance of the Performance Consultant.

A metric-focus grid is used for collecting, communicating, analysing, and presenting performance data. A metric is a time varying function that characterises some aspect of the performance of a parallel program; e.g. CPU utilisation, memory usage and number of floating point operations. Focus is a specification of a part of a program expressed in terms of program resources. For example, one focus might be all synchronisation objects accessed by a single procedure on one processor. A matrix is stored with each metric listed for each program component. The elements of the matrix can be single values (such as current value, average, min, or max) or time-histograms. Time-histograms are fixed-size data structures that record the behavior of a metric as it varies over time.

7.1.3 Pablo Performance Analysis Toolkit

The Pablo Performance Analysis Toolkit, developed by the Pablo group of Department of Computer Science, University of Illinois at Urbana-Champaign, is a portable performance data analysis environment that can be used with a variety of massively parallel systems [82, 90]. The Pablo software environment combines portable performance data analysis with portable source code instrumentation and data capture.

The Pablo Performance Analysis Toolkit consists of several components that provide for the capture and analysis of performance data from serial and parallel programs. All of these components use the Pablo Self-Defining Data Format (SDDF) as their medium of information exchange. Among the major components, there are:

- Trace Library and Extensions: the base performance capture library and extensions for recording timestamped event records and performance information about application constructs of particular interest, for example MPI I/O calls, message passing calls, and I/O requests.
- I/O Analysis: software that produces summaries of I/O activities from I/O event records.
- SvPablo [93], graphical environment for instrumenting applications and browsing dynamic performance data.
- Pablo Analysis GUI, an analysis and visualization GUI that processes SDDF files.

In addition to the above-mentioned components, software is available for capturing and analysing various other activities, including I/O activities at different levels.

Calls to the SvPablo performance instrumentation library are generated to capture dynamic performance data. During execution of the instrumented code, the SvPablo maintains statistics on the execution of each instrumented event on each processor and maps these statistics to constructs in the original source code. Performance browsing using SvPablo involves correlating the performance data gathered at runtime with the application source code and performing statistical analysis. A performance file is also created and stored in the Pablo Self-Defining Data Format (SDDF) and is used by SvPablo as and when required.

The SDDF format allows representing performance data flexibly with optional and configurable fields and thus making it easy to introduce new performance metrics without modifications to the associated SvPablo performance browser.

7.2 Performance Monitoring in Grid Environment

As mentioned earlier, Grid performance analysis is distinctively different from the performance analysis for traditional parallel architectures. In particular, the dynamic nature of Grid, heterogeneity of resources and multiple administrative domains make most parallel and distributed programming paradigms unsuitable and the existing parallel programming and performance analysis tools and techniques are not always appropriate for coping with such an environment.

The fundamental and ultimate goal of high performance computing is speeding up the computation, i.e. executing the same task in shorter time. In a Grid environment, however, speed of the computation is not the only issue. Mapping application processes to the resources in order to fulfill the requirement of the application in terms of power, capacity, quality and availability forms the basis of Grid performance evaluation. Enormous amount of monitoring data are generated in a Grid environment, which are used to perform fault detection, diagnosis and scheduling in addition to performance analysis, prediction and tuning [4, 85].

The important elements that make performance analysis for Grids different from the performance analysis for SMPs or MPPs may be summarised as follows [72]:

- **Lack of prior knowledge about the execution environment:** The execution environment in a Grid is not known beforehand and also the environment may vary from one execution to another execution and even during the same execution of an application.
- **Need for real-time performance analysis:** Due to the dynamic nature of a Grid-based environment, offline analysis is often unsuitable. Sometimes it becomes necessary to predict the behaviour of an application based on some known

characteristics of the environment. However, in the absence of precise knowledge about the future shape of the Grid, this prediction-based analysis is also of little use.

- **Difficulty in performance tuning of applications:** It may not be possible to exactly repeat an execution of a given application in a dynamic Grid environment. Therefore, performance tuning and optimisation of the application is difficult.
- **Emergence of new performance problems:** The very different nature of a Grid infrastructure gives rise to new performance problems that must be identified and tackled by the performance analysis tool.
- **Requirement of new performance metrics:** Usual performance metrics used by traditional performance analysis tools are inappropriate and therefore new performance metrics must be defined.
- **Overheads due to Grid management services:** Grid information services, resource management and security policies in a Grid environment add additional overhead to the execution of an application.

For all the above characteristics of a Grid environment, the performance engineering of an application running on a Grid should be tackled differently. Timely and accurate analysis is important and, as the system is large, complex and widely distributed, it is essential that the monitoring and a substantial part of the analysis activity must be automated.

Global Grid Forum proposed a Grid Monitoring Service Architecture [5, 40], which is sufficiently general and may be adapted in variety of computational environments including Grid, clusters and large compute farms. The essential components of the architecture as described in [5] are as follows:

- *Sensors:* for generation of time-stamped performance monitoring events for hosts, network processes and applications. Error conditions can also be monitored by the sensors.
- *Sensor Manager:* responsible for starting and stopping the sensors.
- *Event Consumers:* request data from sensors.

- *Directory Service:* for publication of the location of all sensors and other event suppliers.
- *Event Suppliers or Producers:* for keeping the sensor directory up-to-date and listening to data requests from event consumers.
- *Event archive:* for archiving the data that may be used later for historical analysis.

The next section presents case studies on three Grid monitoring tools and highlights their important characteristics.

7.3 Performance Monitoring and Analysis Tools for Grid

Currently available Grid Monitoring Tools may be classified into two main categories: Grid infrastructure monitoring tools and Grid application monitoring tools. Although the primary focus of majority of the tools is on monitoring and analysis of Grid infrastructure, many tools are nowadays concentrating on a unified approach. Following subsections present a brief overview of three such tools in order to get some idea about the basic components. A detail survey of existing performance monitoring and evaluation tools may be found in [39] which have an objective of creating a directory of tools for enabling the researchers to find relevant properties, similarities and differences of these tools.

7.3.1 Relational Grid Monitoring Architecture

Relational Grid Monitoring Architecture (R-GMA) [91] was developed within the European Data Grid Project as a Grid Information and Monitoring System. It is based on the GMA from Global Grid Forum. R-GMA exploits the power of the relational model offering a global view of the information. R-GMA implementation creates impression that each Virtual Organisation (VO) in Grid has one large relational database, although it does not quite resemble with a general distributed RDBMS structure. The database is used both for publishing information about the Grid (primarily to find out about what services are available at any one time) and for application monitoring. The information infrastructure is modeled as a set of producers and consumers and a single registry.

The producers and consumers access, publish and collect information via SQL statements. Producers provide a description of information they can offer by using a CREATE TABLE statement and publish the information with an SQL INSERT statement. Each tuple inserted in the tables by a producer contains a timestamp to fulfill the requirements of monitoring system. Consumers use SQL SELECT statements to collect the information they need.

Producers may be of three different classes: *Primary*, *Secondary* and *On-demand* [114]. Each type of producer is created by a user call, but they differ in the way the tuples are inserted in the virtual table. In case of a primary producer, the tuples are inserted periodically by the user code into the internal storage maintained by the producer service. On the other hand, a secondary producer populates its internal storage with tuples obtained from other producers by running its own query. Both, primary and secondary producers, respond to the queries from other user applications or consumers by retrieving data from their internal storages. However, the third type, the on-demand producers do not maintain an internal storage, instead they respond to a query by forwarding it directly to the user code first and then obtaining data from the user code.

A query may be a) *static* responding with static monitoring data, b) *continuous* responding with a continuous stream of tuples immediately after they are inserted into the producer's internal storage, c) *latest* responding with the tuples that represent only the current state and d) *history* responding with all the matching tuples (may be during a specified time interval). The data from the internal storage may periodically be purged after a certain "Retention Period".

7.3.2 SCALEA-G

SCALEA-G [95], developed at the University of Vienna is the next generation of the *SCALEA* System. SCALEA-G is a unified system developed for monitoring Grid infrastructure based on the concept of Grid Monitoring Architecture (GMA) [103] and at the same time for performance analysis of Grid applications [107]. SCALEA-G is implemented as a set of Grid services based on the Open Grid Service architecture (OGSA)

[33]. OGSA-compliant Grid services are deployed for online monitoring and performance analysis of a variety of computational resources, networks, and applications.

The major services of SCALEA-G are: (a) *Directory Service* for publishing and searching information about producers and consumers of performance data, (b) *Archival Service* to store monitored data and performance results, (c) *Sensor Manager Service* to manage sensors. Sensors are used to collect monitoring data for performance analysis. Two types of sensors are used: *system sensors* (for monitoring Grid infrastructure) and *application sensors* (to measure the execution behaviour of Grid applications). XML schemas are used to describe each kind of monitoring data and any client can access the data by submitting Xpath / Xquery-based requests. The interactions between sensors and Sensor Manager Services also take place through the exchange of XML messages.

SCALEA-G supports both source code and dynamic instrumentation for profiling and monitoring events of Grid Applications. The source code instrumentation service of SCALEA-G is based on *SCALEA* Instrumentation System [106]. Dynamic instrumentation (based on Dyninst from [25]) is accomplished by using a *Mutator Service* that controls the instrumentation of application process on the host where the process is running. An XML-based Instrumentation Request Language (IRL) has also been developed for interaction with the client.

7.3.3 NetLogger and JAMM Monitoring System

NetLogger (stands for Networked Application Logger) Toolkit [104], developed at Lawrence Berkeley National Laboratory, University of California, Berkeley, is used for monitoring the behaviour of all the elements of the application-to-application communication path, applications, operating systems, hosts, and networks. The NetLogger Toolkit consists of the following components and tools: an API and library of functions for generation of application-level event logs, tools for collecting and sorting log files, host and network monitoring tools, and a visualization and analysis tool. The NetLogger tools share a common log format.

JAMM, the Java agents for Monitoring and Management is an automated agent-based monitoring system [102], which is often used to collect monitoring events for use with the NetLogger Toolkit. The monitoring agents can be used for monitoring a wide range of system and networking elements and then for extraction and publishing the results. On-demand monitoring reduces the total amount of data collected, which in turn simplifies data management. The components of JAMM include sensors, sensor manager, directory service, event consumers and event gateway (listen requests from event consumers). In addition, event data are archived for historical analysis and performance prediction.

7.3.4 GrADS and Autopilot

Researchers from different universities led by a team at Rice University are working in the GrADS project [7, 90] that aims at providing a framework for development and performance tuning of Grid applications. As a part of this project, a *Program Execution Framework* is being developed [60] to support resource allocation and reallocation based on performance monitoring of resources and applications. A performance model is used to predict the expected behaviour of an application and actual behaviour is captured during its execution from the analysis of measured performance data. If there is a disagreement between the expected behaviour and observed behaviour, actions like replacement or alteration of resources or redistribution of the application tasks are performed.

The monitoring infrastructure of GrADS is based on Autopilot, which monitors and controls applications, as well as resources. The Autopilot toolkit supports adaptive resource management for dynamic applications. Autopilot integrates dynamic performance instrumentation and on-the-fly performance-data reduction with configurable, malleable resource management algorithms and a real-time adaptive control mechanism. Thus, it becomes possible to automatically choose and configure resource management policies based on application request patterns and system performance [92]. Autopilot provides a flexible set of performance sensors, decision procedures, and policy actuators to accomplish adaptive control of applications and resources. The policy implementation mechanism of the tool is based on fuzzy logic.

7.4 Distinctive Characteristics of Performance Analysis Tools

The above discussion brings out the salient features of different performance monitoring and analysis tools for Grid environments and shows how they are different from those for traditional parallel architectures. It may be noted that these tools are only representatives of the vast number of performance analysis tools available for the above mentioned systems. Below the different features of these two sets of tools are summarized for a better understanding.

First the characteristics of performance analysis tools used for traditional parallel architectures are highlighted.

- The important components of performance analysis tools are (a) tools for instrumentation and data collection, (b) tools for analysing and identification of performance problems, and (c) tools for visualisation.
- Monitoring data is collected through source code instrumentation and trace libraries are used for recording monitoring events. For example, *SCALEA* uses SISPROFILING, and PAPI [86] (does profiling using timers, counters, and hardware parameters) libraries. PAPI is also used by the Pablo toolkit along with Autopilot libraries.
- Source code instrumentation is generally guided by the user (in an interactive manner). Modern tools employ dynamic instrumentation techniques to dynamically change the focus and performance bottlenecks are identified during run-time.
- The tools are designed to work on diverse hardware platforms**.** For example, the platform dependent part of *Paradyn* [87] is available for SPARC, x86 and PowerPCs.
- One requirement of these tools is to support different programming paradigms. *SCALEA*, for example, supports performance analysis of HPF, OpenMP and MPI codes.
- Each tool defines their own data storage formats, thus limiting the portability of monitoring data.

However, it is evident that on a large, complex and widely distributed system similar to Grid, many of the above features including post-mortem performance analysis are of no use. Thus, all tools in Grid require to perform real-time analysis. Some existing tools also use this analysis data for automatic performance tuning of applications. The essential ingredients of real-time analysis are

- *dynamic instrumentation and data collection mechanism* (as performance problems must be identified during run-time),
- *data reduction* (as the amount of monitoring data is large and movement of this amount of data through the network is undesirable),
- *low-cost data capturing mechanism* (as overhead due to instrumentation and profiling may contribute to the application performance), and
- *adaptability to heterogeneous environment* (for the heterogeneous nature of Grid, the components along with the communicating messages and data must have minimum dependence on the hardware platform, operating system and programming languages and paradigms).

Many of the already existing libraries and performance monitoring tools for traditional parallel systems form parts of the Grid monitoring tools. For example, SCALEA-G uses the instrumentation system of *SCALEA* [106] and Dyninst [25] and one of the current foci of GrADS project is to incorporate the instrumentation of SvPablo [93]. However, when the tools extend their functionality to incorporate real-time performance analysis and tuning, they have to confront with complex issues like fault diagnosis and application remapping.

The recent trends in Grid performance monitoring and analysis tools are summarized below.

- The basis of most monitoring tools is the general Grid Monitoring Architecture proposed by the Global Grid Forum. The architecture allows much flexibility, as well as scalability in the system.
- Most Grid monitoring tools are built for infrastructure monitoring (e.g. RGMA). However, the recent trend is to combine the infrastructure monitoring with

application monitoring (e.g. SCALEA-G). Attempts have also been made to connect existing parallel application monitoring tools (e.g. GRM) with Grid monitoring tools (e.g. R-GMA) (see [84]).

- The tools provide static and dynamic information about the Grid. Many tools archive the monitoring information and satisfy queries on historical data. Applications are instrumented dynamically for collecting dynamic information about their execution.
- Generally low level monitoring information is presented to the users, for example, CPU and memory usage or start or end of messages etc. This is different from the monitoring tools for the traditional parallel systems, in which high level abstractions, such as overhead-based analysis information are presented.
- Presentation of data is different for different tools. For example, NetLogger uses a self-describing data format to log the events and a visualization component presents the data to the user. On the other hand, R-GMA supports SQL queries for retrieving information according to the needs of the consumers.
- The recent trend for the monitoring tools is to migrate to web services and to enable the users browsing monitoring information through web pages.
- So far very few tools have extended their domain to include performance tuning of applications in Grid environment. The GrADS project attempts resource allocation and reallocation to meet the performance expectation of an application. A performance steering approach [68] has been adopted in the RealityGrid project [89] in which the performance of an application is improved through an adaptive process involving run-time adjustment of factors.

The thesis does not focus on evolving new performance monitoring strategies for Grid environment, although it understands that further research work needs to be pursued in order to develop future generation performance monitoring and analysis tools. The implementation of the resource management framework which is being dealt with in this thesis depends on existing tools for performance monitoring services. Also, only preliminary implementation of performance analysis of jobs (components of an application)

executing on the Grid resources has been incorporated in the PRAGMA tool. The implementation is discussed in the next section.

7.5 Performance Monitoring and Analysis Within PRAGMA

PRAGMA implements the multi-agent system, which has been described in Chapter 4. Among all the agents, the Analysis Agent is employed for the performance analysis activities in PRAGMA. Because performance monitoring and analysis in not in particular focus of this thesis, only a preliminary implementation of the Analysis Agent has been incorporated as discussed here.

Data Collection

In order to gather performance-monitoring data at the time of execution of a job, this work rely on Performance Application Programming Interface (PAPI) [86]. PAPI monitors the runtime behaviors of any on-time executing job. PAPI provides a standard API for accessing hardware counters available on most of the modern microprocessors. These counters exist as a small set of registers that count events, which are occurrences of specific signals related to a processor's function. The PAPI specification consists of a standard set of events and high-level and low-level sets of routines for accessing the counters. The high level interface simply provides the ability to start, stop, and read sets of events, and is intended for the acquisition of simple but accurate measurements. The fully programmable low-level interface provides sophisticated options for controlling the counters, such as setting thresholds for interrupt on overflow. However, the present implementation uses the high-level interface for gathering data and sending that to the *Analysis Agent*. The application codes are suitably instrumented to capture a set of events, data regarding the occurrences of events are stored in a data buffer and Analysis Agent use a *pull model* for obtaining data from the data buffer. This data is then analysed by the Analysis Agent in order to discover the occurrences of any performance problem.

Information obtained from PAPI also includes *System Information* (processor, total memory, system page size), *Processor Information* (vendor, processor family, model, etc.),

Cache (level 1 details and level 2 details) and *TLB Information* (level 1 and 2 details), and *Processor Features* (APIC, CMOV, Cx8, etc.). A Java interface has been built as an upper layer on top of Performance APIs of PAPI to access different cross platform hardware preset or native events. The *Analysis Agent* sits on top of PAPI and the Java interface and detects any performance problem on the basis of PAPI-generated monitoring data. The design layers as implemented in the current version of `PRAGMA` are presented in Figure 7.1.

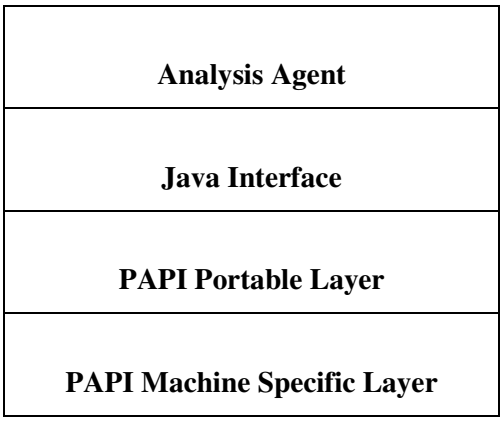


Figure 7.1: Different design layers.

Analysis of Performance Data

The JobExecutionManager Agent sends the SLA of a particular job to the Analysis Agent residing on a particular resource provider on which the job is allotted. The Analysis Agent verifies specific SLA commitments for that particular job by collecting performance data from PAPI at regular intervals. Thus, the following steps are executed by the Analysis Agent.

Step 1: For every resource attribute present in SLA, the Analysis Agent obtains information through Ganglia and PAPI and checks the SLA commitments.

Step 2: Analysis Agent considers specific tolerance for any SLA commitment violation.

Step 3: If the violation is within the tolerance then Analysis Agent recommends to continue the execution.

Otherwise, the Tuning Agent is invoked for specific action.

In this implementation, the Analysis Agent only collects information about number of instructions so far completed (from PAPI) at a particular point of time (denoted by N_p). The time taken by the job to complete N_p number of instructions (denoted by T_{Np}) is also measured using PAPI and is obtained by the Analysis Agent.

The Analysis Agent also obtains the following information from the SLA.

N: the total instruction count of the job.

T_{ect}: expected completion time of the job, which is mentioned in the SLA.

Based on this information, the Analysis Agent estimates the completion time T_c of the job by computing:

$$T_c = T_{Np} + (T_{Np}/N_p * (N - N_p))$$

If $(T_c > T_{ect})$ (i.e. when the estimated completion time is larger than the expected completion time mentioned in the SLA), then the Analysis Agent takes the decision for local tuning or rescheduling depending on the following situations:

i. If the resource provider has more resources (which are not overloaded) than it is currently using for running the job, then the Analysis Agent recommends local tuning to improve the performance of the job by providing additional resources (e.g., processors, memory).
ii. If additional resources are available on the current resource provider but these resources are overloaded by some other computational jobs, then the Analysis Agent recommends for rescheduling to some new resource provider.
iii. If the current resource provider does not have additional resources then the Analysis Agent recommends for rescheduling to a new resource provider with enhanced resources, which can improve the performance of the suffered job.

As mentioned, detailed implementation of the Analysis Agent is not the main focus of this research work. However, a simple approach based on the above procedure has been developed to fulfill the requirement.

7.6 Conclusion

This chapter discussed and highlighted the differing monitoring techniques for traditional parallel architectures and Grids. The objective is to highlight the current trend in research in this area. The observations that surfaced in this chapter help to grow an understanding of the recent road map of performance monitoring tools for large, distributed computer systems. In addition, a direction for new generation tools is also set up.

One important requirement for a dynamic, distributed environment like Grid is to adapt the jobs to the changing resource usage scenario. Two major approaches are adopted to bring such type of adaptability to the jobs. First, when the performance degrades, the job may be tuned locally and may be provided with more resources so that the required QoS is maintained. Secondly, the job may be migrated to a different resource provider in order to maintain the QoS as desired by the client. The second situation may arise when the load on a particular resource provider increases or the resource provider fails to keep up its promise. This thesis mainly focuses on the second aspect of adaptive execution and discusses in detail its implementation within PRAGMA in Chapter 8 of this thesis. As mobile agents are used as the major tool for providing adaptive execution facility, next chapter first presents a brief study on the existing mobile agent systems. The later part of Chapter 8 presents the detailed design of PRAGMA for providing adaptive execution facility using mobile agents.

Chapter 8

Adaptive Execution of Concurrent Jobs

Probably, one of the most challenging problems that the Grid computing community has to deal with is the fact that Grid is a highly dynamic and complex environment. The multi-agent system described in Chapter 4 provides support for adaptive execution in such a dynamic environment. This adaptive execution facility is availed by reallocation and local tuning of jobs if performance degrades because of changes in the availability of Grid resources or if the users' QoS requirement is not maintained.

This chapter first introduces the adaptive execution ability of the multi-agent system for applications running in a Grid environment. It also highlights the importance of mobile agents in providing adaptive execution feature specifically in rescheduling. For this purpose, some of the existing mobile agent systems are also briefly discussed in this chapter. This chapter also explains the implementation of adaptive execution features in `PRAGMA`, which offers rescheduling, as well as local tuning services.

8.1 Adaptive Execution

Chapter 4 presents the design of a multi-agent system (MAS) for handling the execution of multiple concurrent jobs. Among all the agents within the MAS, the JobController Agent and a set of JobExecutionManager Agents deal with the initial scheduling of all the concurrent jobs. Once a resource provider is selected for a job and the job is submitted onto it, the JobExecutionManager Agent moves to that selected resource provider along with the

job. Thus, the JobExecutionManager Agents become associated with the jobs and all of them are actually supervised by the JobController Agent. An Analysis Agent sitting on a particular resource provider monitors the execution of a job submitted to that resource provider in addition to monitoring the execution environment. After initial allocation of a job, the JobExecutionManager Agent associated with it along with the Analysis Agent and Tuning Agent on the selected resource provider manages the adaptive execution of the job.

The JobExecutionManager Agents are implemented as mobile agents. If the resource provider fails to maintain its promised QoS (defined in the SLA) and the performance of the job degrades, then in order to improve the performance of the suffered job, Analysis Agent suggests either rescheduling of the job to some new resource provider or initiates local tuning of the job on the current resource provider. Analysis Agent decides the next course of action after consulting the SLA which it receives from the JobController Agent. The expected completion time (T_{ect}) of a job is mentioned in the SLA (see Section 5.6). Analysis Agent monitors job performance in periodic interval to ensure that T_{ect} will be achieved. For example, Analysis Agent computes estimated completion time (T_c) of the job (see Section 7.5) and checks whether the following condition is satisfied:

$$T_c \leq T_{ect}$$

If the above condition is not satisfied, Analysis Agent either decides to dynamically tune its performance on the same resource provider or to reschedule it onto a different resource provider. If the decision is to tune the job performance locally, the Analysis Agent also checks whether the current resource provider can enhance the resources utilized by the job. In this context, Analysis Agent considers four situations, which are discussed below:

- *Providing additional resources:* If the resource provider has more resources (e.g., 4 processors) than it is currently using for running the job (e.g., 2 processors), then Analysis Agent suggests to improve the performance of the job by providing additional resources. On the basis of this suggestion, Tuning Agent performs local tuning of the job such that it can access these additional resources.

- *Changing the scheduling strategy:* The Analysis Agent may also suggest some run-time modification of the code, such as changing the scheduling strategy of the parallel part of the code in case of load imbalance.
- *Rescheduling to a relatively idle resource provider:* If additional resources (e.g., processors) are available on the current resource provider but are overloaded by some other computational jobs, then the Analysis Agent suggests for rescheduling to some new resource provider (see Section 5.5).
- *Rescheduling to a resource provider with higher capability:* In another situation, if the current resource provider does not have additional resources then also the Analysis Agent suggests for rescheduling to a new resource provider with enhanced resources, which can improve the performance of the suffered job. The JobExecutionManager Agent selects a new suitable resource provider and reschedules the job to this newly selected resource provider (see Section 5.5).

While scheduling and rescheduling, mobile agents are deployed for carrying the jobs to the selected resource providers. In order to provide this facility, a mobile agent needs an environment at the remote resource provider. This facility can be provided by creating mobile agent stations on the remote resource providers and these stations register themselves in one or more lookup services. Mobile agents search for a particular mobile agent station from these lookup services, Figure 8.1 demonstrates the implementation model of mobile agent using lookup services and mobile agent stations.

The next section discusses some Java based mobile agent systems, which could be used to design a mobile agent framework for PRAGMA. Section 8.3 discusses how Jini can be used to implement agent-based systems including mobile agents and explains the rationale behind considering Jini for this purpose. Section 8.4 discusses the implementation details of job migration in PRAGMA and presents a performance model of this implementation. Although, local tuning techniques have not received much attention in this thesis, Section 8.5 discusses some simple local tuning techniques used by PRAGMA in order to improve the performance of a job while executing on the same resource provider.

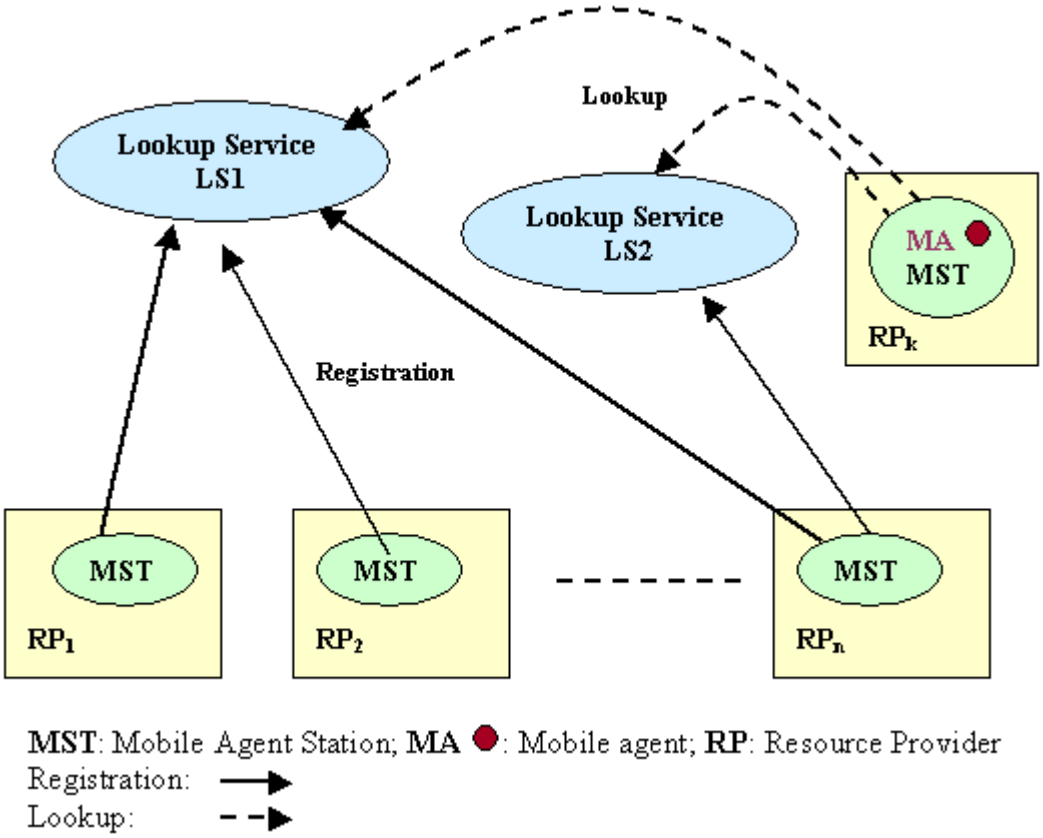


Figure 8.1: Lookup services, mobile agent stations and mobile agent.

8.2 Java based Mobile Agent Systems

There are several Java based mobile agent systems available [116]. These mobile agent toolkits provide all needed classes in Java. Here is an overview of some Java-based agent systems: Aglets, Concordia, Naplet, Odyssey, and Voyager. These Java based mobile agent systems share certain characteristics: they provide an agent server, the agents can migrate from one server to another server, carrying their state with them, and the agents can load their code from variety of sources.

8.2.1 Aglets

Aglets Software Development Kit (ASDK) [2] is a product of IBM's Tokyo Research Laboratory, initiated in early 1995. The goal has been to bring the flavor of mobility to the applets (Aglet means agent plus applet) and to build a network of Aglets hosts with the task

specific Aglets. An agent is called an aglet in this system. The Aglets framework employs a weak migration mechanism that transfers only agent program and data. Tahiti is an application program that runs as an agent server or aglet server. It provides a user interface for monitoring, creating, dispatching, and disposing of agents. Aglets can approach only the systems that are running the Aglets Server. The Aglet architecture consists of the Aglet API (set of Java classes and interfaces that allows creation of mobile agents), the Aglets Runtime Layer and two implementation Layers - the Agent Transport and Communication Interface (ATCI) and the Agent Transfer Protocol (ATP). The Aglets Runtime Layer is an implementation of the Aglets API that provides the fundamental functionality for Aglets to be created, managed and dispatched to remote hosts. Aglet system uses socket mechanism for migration. At the time of migration, an Aglet sends a request to the Aglet Runtime Layer. The Layer converts the Aglet by serialization into the form of byte array consisting of its data and code. The resulting byte stream is passed to the Agent Transfer Protocol (ATP) through Agent Transport and Communication Interface (ATCI) that makes it protocol-independent. The ATP constructs a bit array containing general information about the Aglet system and Aglets identification together with information given from the Aglets Runtime, after which the Aglet is ready for transfer.

8.2.2 Concordia

Concordia [19] is a mobile agent system developed by Mitsubishi Electric ITCA (MEITCA). Concordia is a framework for development and managing network-efficient mobile agent applications for accessing information anytime, anywhere and on any device. Concordia supports weak migration only. Concordia has the Concordia server that includes several modular components that work together to provide an integrated development environment and management tool for its agents. Concordia server comprises of some major components like: Agent manager, Administration manager, Security manager, Persistence manager, Event manager, Queue manager, Directory manager, Service bridge, and Agent tools library. Here is a brief description about each component.

- **Agent manager:** The Agent manager provides an environment for agent's execution and also manages creation and destruction of the agent.

- **Administration manager:** One Administration manager is required for the entire Concordia network; it permits remote administration of Concordia services running on other nodes.
- **Security manager:** The Security Manager is responsible for identifying users, authenticating agents, protecting server resources, and authorizing the use of dynamically loaded classes.
- **Persistence Manager:** The Persistence Manager controls the persistence and recovery of the agents after system or network failure.
- **Event Manager:** The Event Manager handles the Concordia agent collaboration.
- **Queue Manager:** The Queue Manager schedules and reschedules the transport of the agent across the network.
- **Directory Manager:** The Directory Manager provides naming service.
- **Service Bridge:** The Service Bridge provides the interface from Concordia agents to the services available at the various nodes in the Concordia network.
- **Agent Tools Library:** The Agent Tools Library provides all the classes required to build Concordia mobile agents including the agent class itself.

When an agent is launched, the Itinerary Set describes its travel plan. The Itinerary Set is a separate data structure stored and maintained outside the agent. Itinerary contains a list of destinations that details the location (i.e., a hostname of a machine) where the agent is to move and the job it has to do. For example if an agent's itinerary consists of two locations (e.g., M1 and M2) and two jobs (e.g., J1 and J2), then the agent will first go to machine M1 and execute job J1, then move to machine M2 and execute job J2. During the transfer between two hosts, Concordia transports agent code, data, and state information using Java RMI through the Concordia servers and uses its itinerary to determine the next destination.

8.2.3 Naplet

The Naplet system [118] is an experimental framework in support of Java-compliant mobile-agent for network centric distributed applications. The Naplet system is based on weak migration. It provides constructs for agent declaration, confined agent execution

environments, and mechanisms for agent monitoring, control, and communication. The Naplet system consists of two objects: Naplet and NapletServer. An agent is called a Naplet in this system that defines job to be performed on the servers and itineraries to be followed by the agent. The NapletServer is a class that implements a dock of Naplets within a Java virtual machine. It executes Naplets in confined environments and makes host resources available to them in a controlled manner. The NapletServers are run autonomously and cooperatively to form an agent flow space for the Naplets. NapletServer comprises seven major components: Naplet monitor, security manager, resource manager, Naplet manager, messenger, navigator and locator. The Naplet manager provides application programs with an interface to launch Naplets, monitors their execution states and controls their behaviors. The messenger and locator components are used for Naplet tracking and communication. The Naplet system presents agent migration in the form of launching process. The navigator consults the security manager whenever it receives a request for agent migration from the Naplet manager. Then it contacts its counterpart in the destination NapletServer for landing permission. After that the destination navigator consults the security manager and resource manager to determine if the landing permission should be granted. On the arrival of a Naplet, the navigator reports the arrival event to the Naplet manager and then passes the control over the Naplet to the Naplet monitor.

8.2.4 Odyssey

The Odyssey system [77] is a Java mobile agent tool developed by General Magic. It provides a set of Java class libraries for developing distributed mobile applications. The Odyssey system consists of agents, agent system and places. The agent system is a platform that can create, manage, interpret, execute, transfer, and terminate agents. It consists of a set of Java classes to support Odyssey agents and Odyssey places. The place is an environment within an agent system in which agents execute. A network of computers forms a place in the odyssey system. One host can provide more than one place and it is an interface between the agents and host's system resources. Agents in the Odyssey system are created with the help of the Odyssey Agent class. Each agent has its own thread of execution and it can perform tasks on its own initiative. Two types of agents are available in this system:

real agents and workers. A real agent in the Odyssey system can move during its execution and it is not bound to the system where it is created. On the other hand a worker is structured as a set of tasks, each to be performed at specific hosts. At each destination, the worker completes the corresponding task from its list and then it moves to the new host and performs the next task and so on. Odyssey supports weak migration mechanism using Java RMI, which is very useful for real agent implementation, as the real agents may need to continue their tasks on the new node. But in case of workers, their tasks are completed on the local host before they move.

8.2.5 Voyager

Voyager [110] is based on the concept of mobile objects developed by ObjectSpace. This is also a Java based system and supports both traditional client server architecture and agent-based architecture and use weak migration policy by which the mobile agent only takes the executed codes and the states of data. Objects are the basic building blocks of the Voyager system, these objects resides and execute in Voyager applications. Voyager is not built on top of Java RMI. Voyager supports a method for using remote objects similar to Java RMI. However, it is implemented differently and additionally supports mobility. Each class that is to be used as a remote object has a Java interface associated with it. Each method that has the capability of being called remotely needs to be included in the interface. It uses a unique concept, i.e. all the serializable objects (Java source code or class file) can be mobile by using Virtual Code Compiler (*vcc*). The *vcc* utility reads a .class or .java file and generates a new remote enabled "virtual class". The new virtual class contains a superset of the original class functions and allows function calls and message passing even when objects are remote or moving. Voyager allows an object to communicate with an instance of a remote enabled class via a special kind of object called a virtual reference.

8.2.6 Implementing the Multi-Agent System

The multi-agent system presented in Chapter 4 is designed for a specific purpose and all the agents within the system have predefined roles. Implementation of this system requires to support both static and mobile agents. The above mentioned agent systems have been found

to be unsuitable and cumbersome for such type of implementation. Many difficulties could be faced if one of these existing agent systems were picked up for this purpose. Few of such difficulties are discussed here. In Aglet system, an Aglet can load a remote class on demand from a remote Tahiti server, which is the agent execution environment of Aglets. For this purpose, an Aglet must establish an additional network connection with the remote place and this is undesirable for the implementation of the multi-agent system. In reality, agents are sometimes not sustained, which is also a problem with Aglet system. All the above agent systems use different mechanisms to maintain agent's itinerary information in a predefined format. However, agent's itinerary within the multi-agent system for Grid resource management is not predefined and based on some dynamic information of the environment. Thus, it is difficult to manage in the above mentioned agent systems. For inter-agent communication, the agent systems rely on widely different mechanisms. For example, Voyager employs standard Java serialization to transport messages across the network. An Aglet uses remote-agent's proxy that serves as message gateway for the Aglet. On the other hand, Concordia uses events to implement inter-agent communication. However, the proposed multi-agent system is intended to use XML for agent communication. Complexity and large amount of overheads involved with these systems are other issues that must be taken into account. For all the above reasons this research work develops its own agent-based system based on Jini framework.

8.3 Implementing Multi-Agent System using Jini

The basic need of a multi-agent system is a special agent platform or some kind of run-time environment. Chapter 4 already discussed that Java is an excellent choice for platform independence and a good foundation for developing multi-agent systems. The multi-agent system presented in this thesis consists of both stationary agents and mobile agents. Although several agent frameworks have been developed in Java (as discussed in Section 8.2), the existing agent systems are not capable enough to fulfill all the requirements for the multi-agent system. Some complexities are involved in these agent systems (as discussed in the previous section), which must be avoided and further flexibility must be sought for in order to implement the multi-agent system. This encourages developing a Jini-based multi-

agent system that also has the benefits of simplicity and incurs lower overhead than other systems.

For the implementation of mobile agent part, the system needs an infrastructure which tackles the requisite number of mobile agent stations, mobile agent and lookup service as discussed in Section 8.1. Jini incorporates in Java a standardized implementation of several components necessary for multi-agent systems, such as distributed services, distributed events, and more importantly lookup and discovery services. Jini also provides the facility of adding new agents in the system without hampering the existing parts of the system. A new agent could even be added at runtime without the need to restart the complete system, thus increasing the functionality and scalability features of the system. Agents can interact with each other within the Jini network through the request/response framework or through some protocol like XML, KQML. Figure 8.2 demonstrates the interactions among mobile agents and stationary agents using XML document. Jini technology attempts to simplify interactions on a network [57, 101]. The Jini architecture is built upon a fully object-oriented approach to network programming. Jini uses Java Remote Method Invocation (RMI) for this purpose. Some of the capabilities of Jini are discussed below, which have been used to implement a part of the multi-agent system.

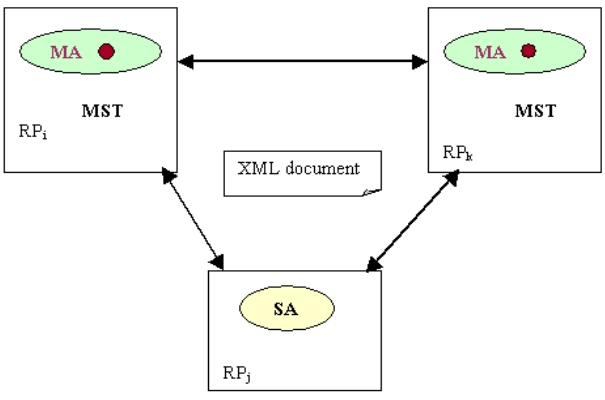


Figure 8.2: Interactions among mobile agents and stationary agents using XML.

Jini technology operates around three protocols that enable discovering and registering services and joining them in *federations* of Jini services [57]. A Jini-enabled service uses the ***discovery*** protocol to announce its presence on a network as shown in Figure 8.3. This is done by *multicasting* a message, i.e. sending information to a number of other machines that belong to the same sub-network simultaneously. The message contains basic information about the service and details about how it can be contacted. Dedicated *lookup services* listen to such announcements, and when they receive one, they contact the announcing service and retrieve detailed information about the service as well as the specific software code that provides an implementation of the service interface (the service proxy) as shown in Figure 8.4. This process is defined in the ***join*** protocol. Finally, other services use the ***lookup*** protocol to discover the lookup services and query them about the registered services as shown in Figure 8.5. At the time of querying the lookup services about the registered services, clients specify the Java types of the service proxy interface implementations that have been uploaded to the lookup services or the service attributes or both. Service attributes are defined using Jini *Entrie*s, where entries are typed sets of objects that can be tested for exact match with a template. Once a required service is discovered, the interface implementation is downloaded to the client and the client can then access the service using Java RMI as shown in Figure 8.6. The use of a service in general can be controlled via the Jini *leasing mechanism* that allows *leases* to be defined on the use of services. The leases must be renewed as and when required, otherwise the services may become unavailable. This is especially useful in case of lookup services, which require that all other services registering with them renew their leases with the lookup services (or define indefinite leases). Through this mechanism Jini ensures that if other services are not available for some reason they will eventually be removed from the registry of the lookup services. Other significant services available through the Jini technology toolkit include Jini *Distributed Event*s, which allow events that take place on one Jini service be propagated to listeners across the network and Jini *Transaction*s, which allow groups of operations be considered as unique atomic transactions [57,101].

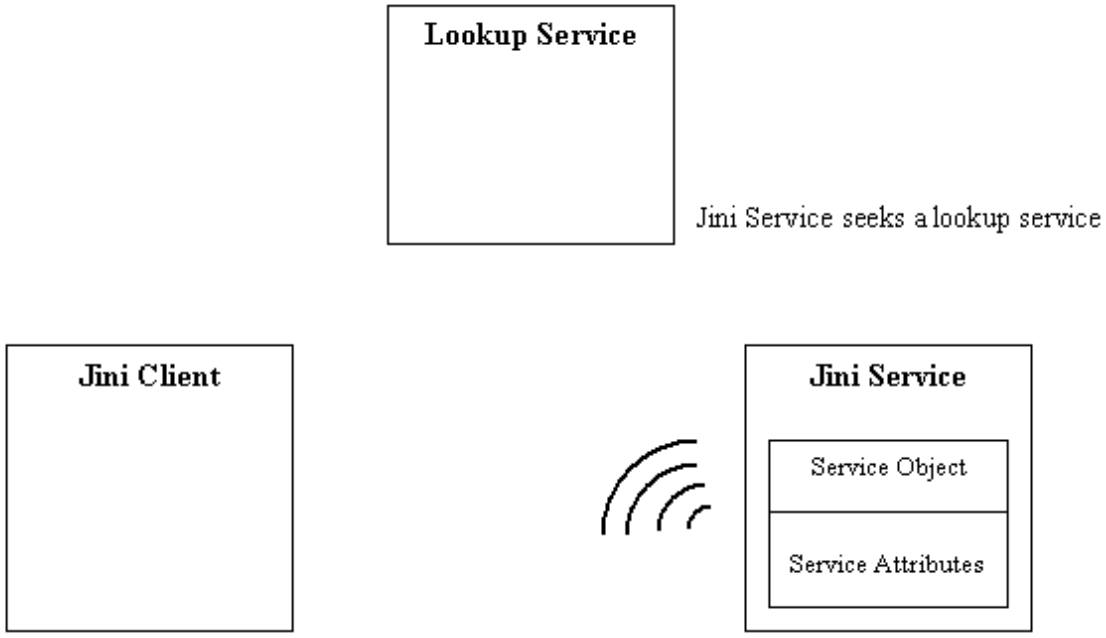


Figure 8.3: Discovery [57].

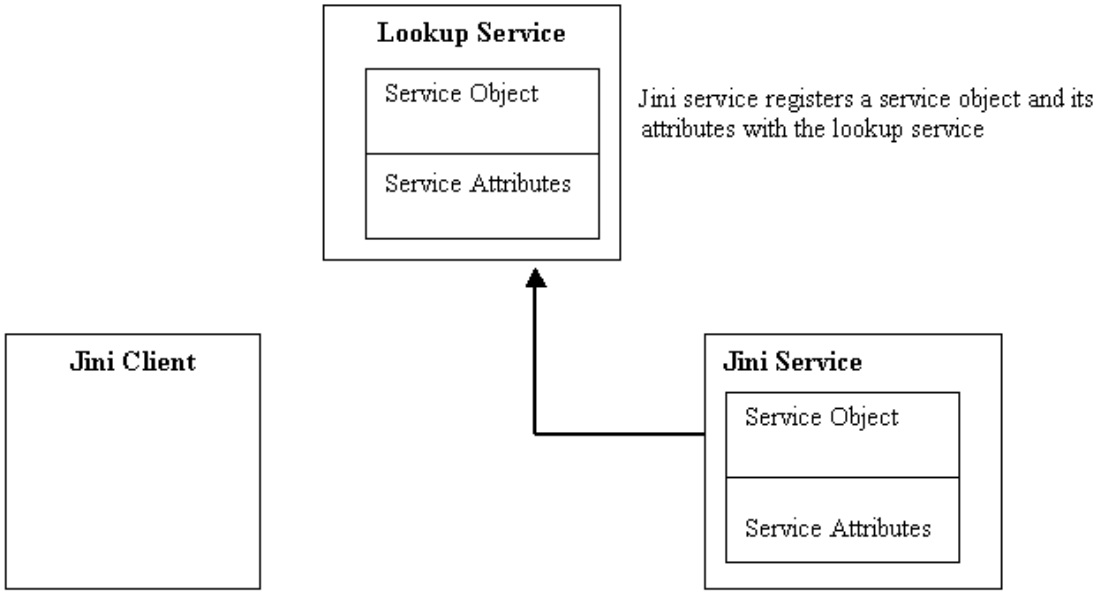


Figure 8.4: Join [57].

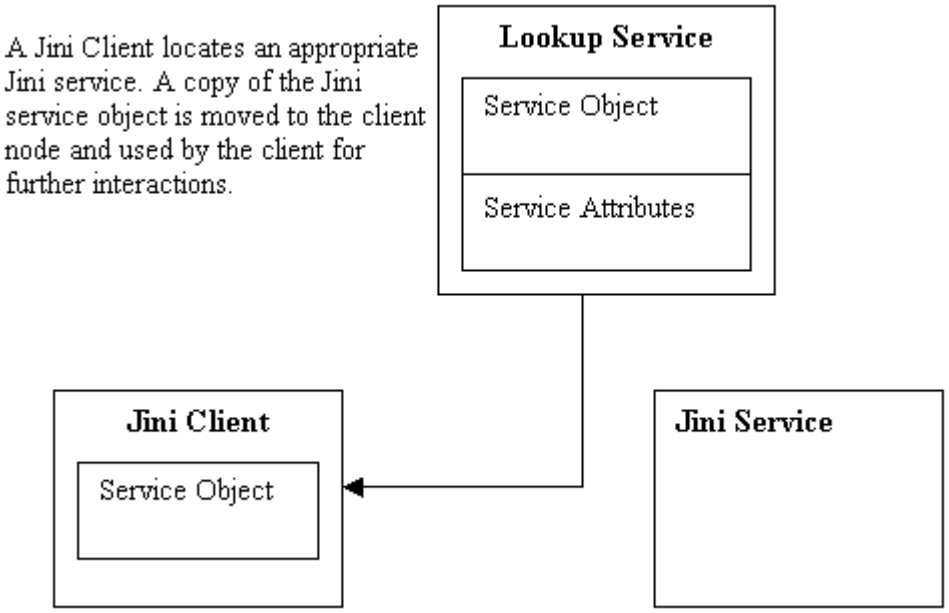


Figure 8.5: Lookup [57].

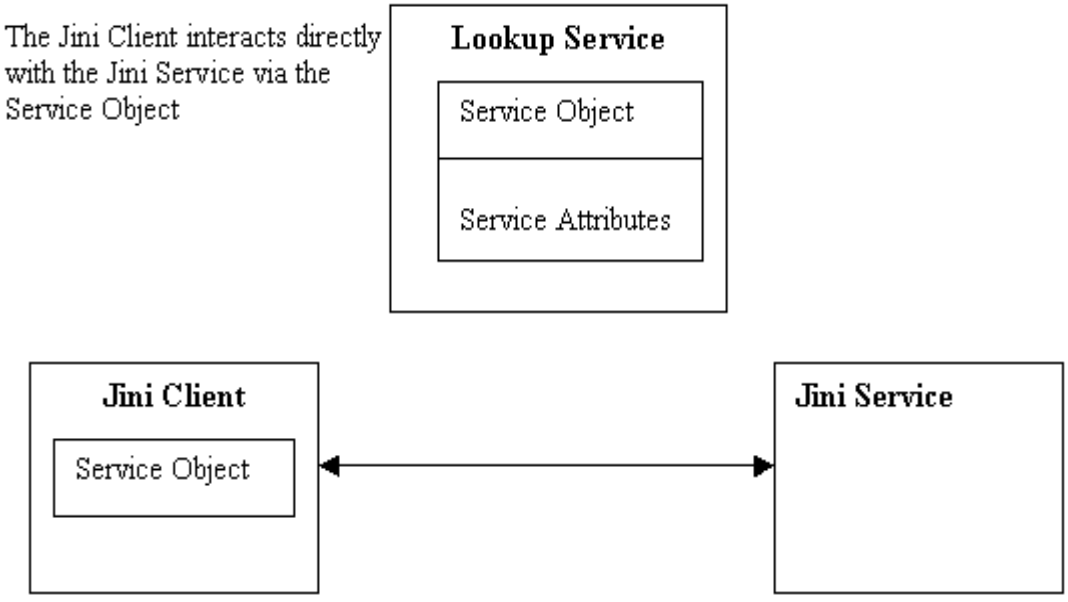


Figure 8.6: Jini Service accessing [57].

An agent-based system may be implemented using Jini by allowing all communications to be done through RMI and one needs to download a proxy (the relevant interface implementation) to communicate with other agents. The downside of using Java RMI for this purpose is that it complicates communication between agents that are not located within the same subnet. Byassee in his article "Unleash mobile agents using Jini" [13] discusses in detail how mobile agents can be implemented on a Jini platform. A similar approach has been adopted in the work presented in this thesis. In this research work, implementation of a part of the `PRAGMA` tool, in particular implementation of the JobExecutionManager Agents has been carried out using Jini mobile agents. Mobile agents are employed here for executing the client's job at remote nodes.

The mobile agent component consists of two main entities: *mobile agent* and *mobile agent station*. The former is the mobile agent itself, that is, an entity with some job to do. The latter is the agent station, a service that provides the mobile agents' execution platform. To be an active agent platform, a given node in the system must have at least one active agent station. These two entities can be easily mapped onto the Jini model, described in this section. Jini, at the highest level, provides the infrastructure that enables clients to discover and use various services. In the context of the mobile agent component, the agent station may be offered as a Jini service [13] and the mobile agent becomes the Jini client. Figure 8.7 shows a sequence diagram depicting the interactions between a mobile agent, mobile agent station and lookup service in Jini.

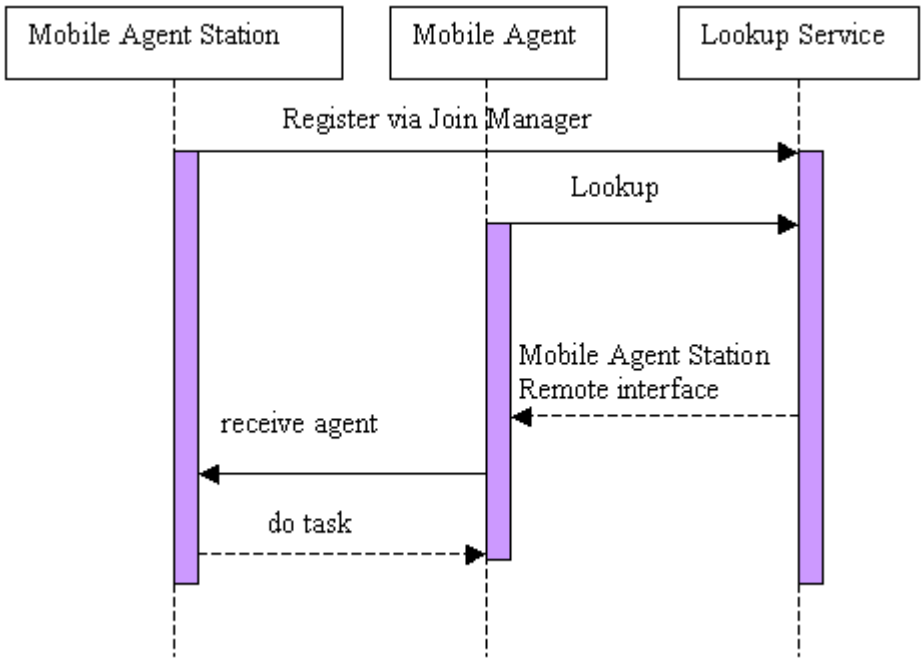


Figure 8.7: Mobile Agent, Mobile Agent Station and Lookup Service in Jini.

8.4 Mobile Agent-based Adaptive Execution

This section describes the implementation of JobExecutionManager Agents and demonstrates how job execution is adapted to dynamic resource conditions and application demands. Adaptation is achieved here by supporting automatic migration of a mobile agent carrying the job in case of performance degradation or resource failure (or overloading). The JobExecutionManager Agent is an initiator of a mobile agent, which carries one of the concurrent jobs to the Grid resources along with the SLA and ResourceProviderList (see Chapter5). This part of the multi-agent system is implemented in `PRAGMA` using Jini technology. The implementation will be described in the remaining part of this section.

`PRAGMA_MobileAgentStation` is a class, which implements a platform for mobile agents on top of a Java virtual machine. It executes mobile agents and makes host resources available to them. Resource providers that are willing to offer computational services start mobile agent stations (one mobile agent station per resource provider) following some

direction from the PRAGMA_JobControllerAgent. The agent stations are run autonomously and cooperatively to form an agent space. Mobile agents move within this space from one station to another to perform their tasks on behalf of their creators. Each mobile agent has a home station in the space. Mobile agent stations register with one or more Jini lookup services by providing a service proxy. In turn, clients, i.e. mobile agents query the lookup services for mobile agent stations that might be of interest.

PRAGMA_JobExecutionManagerAgent (Figure 8.8) initiates a mobile agent to perform a job (one of the multiple concurrent jobs) on the selected resource provider. The mobile agent executes the job on that mobile agent station. During the execution of the job, if PRAGMA_AnalysisAgent detects any performance problem, a warning is sent to the PRAGMA_JobExecutionManagerAgent and at the same time the value of a performance variable *Perf_v* is set. *Perf_v* indicates whether rescheduling is necessary or not (the variable may also be set with other values to indicate other types of actions, such as local tuning of the job). The mobile agent monitors *Perf_v* and, if required, it will stop execution on that mobile agent station and will move to another mobile agent station as indicated by the PRAGMA_JobExecutionManagerAgent.

In order to construct a mobile agent station, a remote interface, which is essentially a service template, is required. MobileAgentStationRemoteInterface has been created for this purpose. The mobile agents actually look for this service template via the Jini lookup service. The MobileAgentStationRemoteInterface provides receiveAgent() method, which a mobile agent calls to move to the implementing mobile agent station. The MobileAgentStationRemoteInterface also provides a method called monitor(), which monitors the performance variable *Perf_v* (set by PRAGMA_AnalysisAgent). An implementation of this remote interface is the actual Jini service. The MobileAgentStation class implements the MobileAgentStationRemoteInterface. The MobileAgentStation constructor first creates a LookupDiscoveryManager to locate the Jini lookup service, and then it creates a JoinManager. The JoinManager simplifies service management

by encapsulating the functionality required for both registering with and withdrawing from a dynamic lookup service pool. The `MobileAgentStation` binds an incoming agent to a thread and subsequently executes (invokes own `run()` method). It also initiates a thread that checks the performance variable *Perf_v* in order to determine whether rescheduling is necessary or not. If rescheduling is required the execution of the agent will stop. `PRAGMA_JobExecutionManagerAgent` then selects a new mobile agent station and moves the mobile agent to that station. The method `newMobileAgentStation()` (discussed later) selects a new mobile agent station.

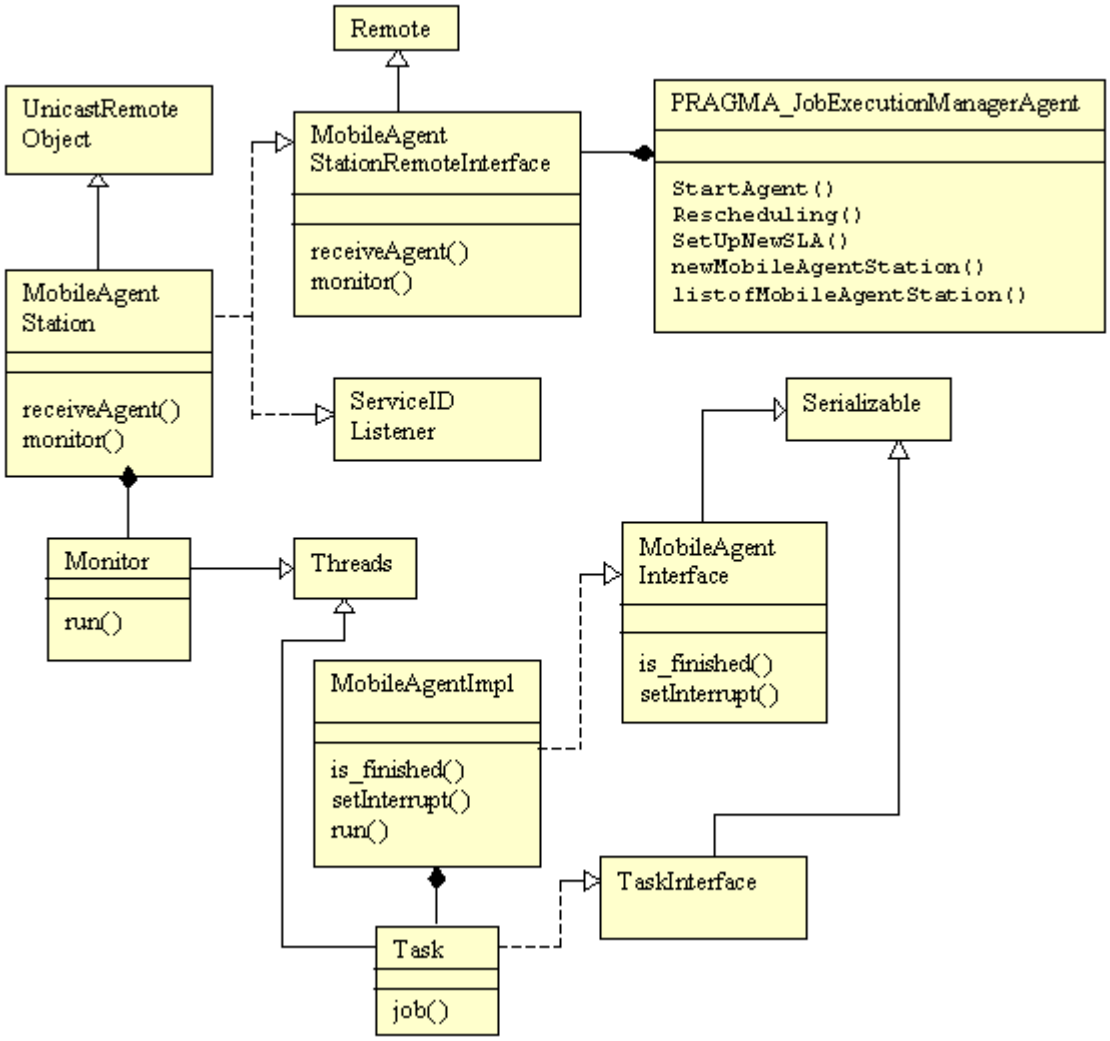


Figure 8.8: Implementation of `PRAGMA_JobExecutionManagerAgent`.

An interface for agents is required for implementing the mobile agent. A `MobileAgentInterface` is created that extends the Java `Serializable` interface. The `Serializable` interface makes the implementer of the `MobileAgentInterface` a serializable entity. `PRAGMA_JobExecutionManagerAgent` moves to a new mobile agent station by calling the `receiveAgent()` method of `MobileAgentStation`. The `StartAgent()` method of `PRAGMA_JobExecutionManagerAgent` is called by `MobileAgentStation` when it arrives to a new station. The `MobileAgentInterface` consists of two methods - `is_finished()` to check whether the mobile agent finishes its task and `setInterrupt()` to stop the execution of the mobile agent as and when required. After being shifted to another mobile agent station the mobile agent can restart its execution there by invoking its own `run()` method.

`MobileAgentImpl` class provides an implementation for the `MobileAgentInterface` class. When the agent needs to move, `PRAGMA_JobExecutionManagerAgent` requires to find an available mobile agent station and invokes `newMobileAgentStation()` method for this purpose. The method `newMobileAgentStation()` checks the list of available mobile agent stations and selects one mobile agent station from that list. To obtain the list of currently available mobile agent station it calls `listofMobileAgentStation()` method. Resource providers that can fulfill the requirements of the job and are selected by the resource broker (collected in the form of ResourceProviderList) register themselves as mobile agent stations. The method `listofMobileAgentStation()` retrieves a list of all current mobile agent stations registered with lookup service and one of them is selected as the new execution platform (following the algorithm presented in Section 5.5). When the agent arrives at the new mobile agent station, the `job()` method in the `MobileAgentImpl` class is invoked and the execution of the job starts. After completion of the execution, the mobile agent returns back to the initial instance of the

PRAGMA_JobExecutionManagerAgent. Figure 8.9 shows the sequence diagram for implementation of the PRAGMA_JobExecutionManagerAgent.

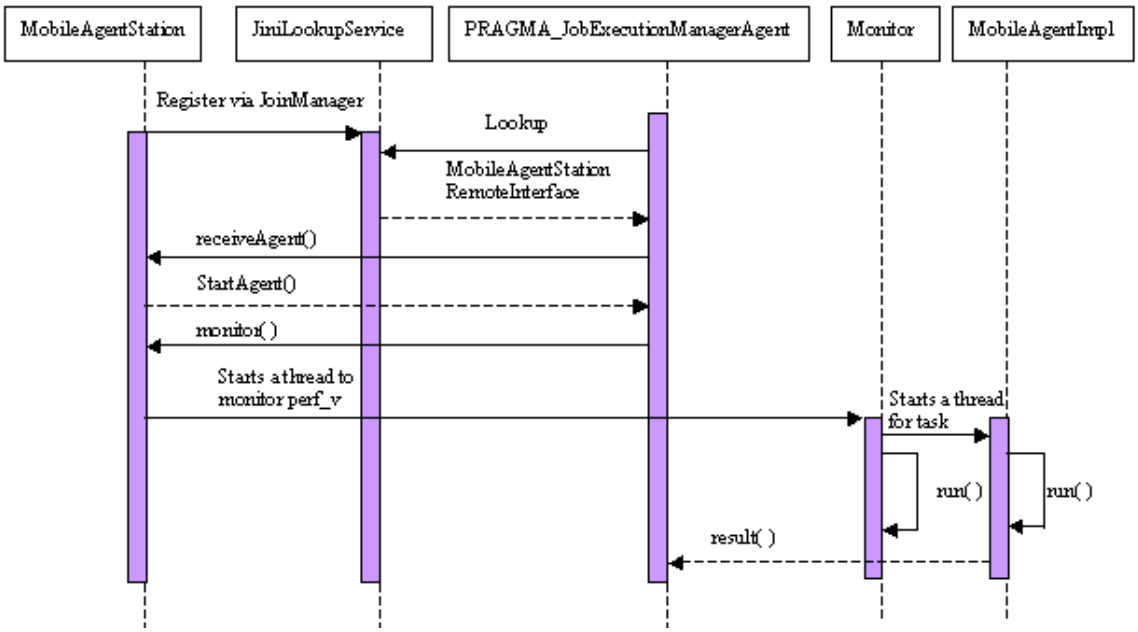


Figure 8.9: Sequence diagram for PRAGMA_JobExecutionManagerAgent.

Figure 8.10 depicts the movement of mobile agent from one mobile agent station to another. Initially, a suitable resource provider for the job is selected by the PRAGMA_JobControllerAgent, later the PRAGMA_JobExecutionManager-Agent picks up the responsibility (according to the figure both are sitting on the client, although they may be located on different Grid resources as well). Thus, the PRAGMA_JobExecutionManagerAgent decides the next mobile agent station when required.

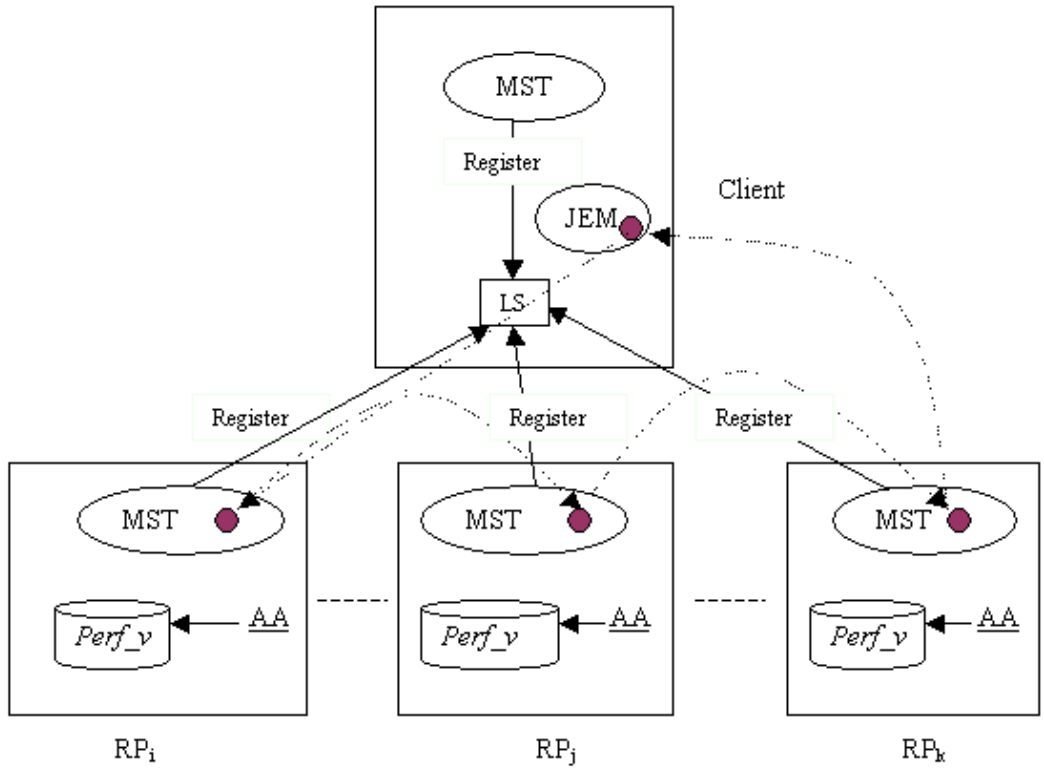


Figure 8.10: Movement of mobile agent from one mobile agent station to another.

Performance

PRAGMA deploys mobile agents for execution of jobs on different resources. The main objective of rescheduling jobs using mobile agents is to improve the performance. Thus, performance of mobile agents significantly affects the overall performance of the jobs. A performance model for interaction of Jini-based mobile agents within PRAGMA is discussed in the remaining part of this section. The discussion extends the concepts presented in [100] and makes it suitable for using within PRAGMA.

Jini interaction model for implementing mobile agents is based on Java Remote Method Invocation (RMI). Java RMI is an object-oriented version of Remote Procedure Call (RPC), which has been widely used as interaction mechanism in distributed systems. RMI provides a simple and powerful Java-to-Java communication model for invoking member

functions on objects that exist on other Java virtual machines. RMI implementation is essentially made through object serialization in Java. The state of an object is converted to a byte stream through the process of serialization. An object that needs to be supplied as an argument to the remote method is serialized by the sending machine and then transmitted. The receiving machine deserializes the object and uses it. The performance model therefore concentrates on the communication dependent part of the RMI. RMI includes binding to the server, marshalling, transfer, unmarshalling of the request parameters, execution of the request, and marshalling, transfer and unmarshalling of the reply. Marshalling is dependent on the size of the request parameters only.

In this implementation, the mobile agent executes on a virtual machine, which is a mobile agent station. A mobile agent station provides an entry point, which allows agents to be received, and an interface, which allows agents to move to another mobile agent station. An object that implements the *Serializable* interface can be saved and restored by the serialization facilities. During each move, the mobile agent along with the client's job (referred as a task hereafter) are serialized and supplied as argument to the remote method at the new mobile agent station. It is then deserialized, i.e. all of these objects and their references are restored.

The network load B_{MA}(in bytes) for a simple mobile agent that moves using RMI from one mobile agent station to another mobile agent station can be estimated as the sum of the size of the serialized mobile agent object MA^s (denoted by $B(MA^s)$) and the size of the serialized task object J^s (denoted by $B(J^s)$).

$$B_{MA}(MA^s,T^s) = B(MA^s) + B(J^s)$$

The execution time T_{MA} for the agent moving from one mobile agent station to another mobile agent station with single hop can be estimated as the summation of the time for binding to the server, T_b, the time for serializing and deserializing the objects (with a factor μ), the time for transferring the data on a network with throughput τ and delay T_δ and the time required to execute the job on the mobile agent stations. Thus, when a mobile agent

moves from a mobile agent station M1 to another mobile agent station M2 for executing a certain job, the total execution time is given by,

$$T_{MA}= 2T_b +2T_\delta + (1/\tau+2\mu)B_{MA}(MA^s,J^s)+T_J^{M1}+ T_J^{M2}$$

T_J^{M1} and T_J^{M1} are the execution times of the client's job on the mobile agent stations M1 and M2 respectively.

Actually in certain circumstances, a mobile agent may need to traverse multiple hops. In such cases the execution time T_{MA} becomes

$$T_{MA}= (n+1)T_b +(n+1)T_\delta+(n+1)(1/\tau+\mu)\ B_{MA}(MA^s,J^s)+ \sum_{i=1}^{n+1} T_J^{Mi}$$

n is the number of hops.

T_b is the time for binding to the server (multiplied with n+1, because of n hops there must be n+1 mobile agent station).

T_δ is the delay, (multiplied with n+1, i.e., nT_δ delay for n hops plus T_δ delay for returning back to the node from where the mobile agent has actually originated).

$(n+1)\mu\ B_{MA}(MA^s,T^s)$ is the time for serializing and deserializing the mobile agent and the client's job J on each mobile agent station during n hops (including at the time of returning back to the first node).

τ is the network throughput (amount of bytes transferred per unit amount of time from one mobile agent station to another station).

T_J^{Mi} is the execution time of the job J on a mobile agent station Mi.

In the above expression, it is assumed that the nodes are connected via a network with uniform delay and throughput. However, as this condition may not always be true, the equation needs to be modified in order to model a situation in which T_δ and τ vary.

8.5 Local Tuning

As mentioned in Section 8.1, if the job running on a particular resource provider can be tuned to more effectively use the current resource provider, Analysis Agent may suggest for application of some local tuning actions to improve the performance of the job. In fact,

an Analysis Agent first checks whether the performance can be improved by tuning a part of the job or by providing additional resources on the same resource provider before taking any decision regarding rescheduling. Thus, local tuning is the first option that is exercised for improving performance of the job. However, like Analysis Agents, Tuning Agents are also not within the direct focus of this thesis. This section presents a brief discussion on the probable implementation of local tuning activities. Only a simple approach is presented and the result of this implementation is demonstrated in the next chapter.

In a Grid environment performance analysis must be done at run-time and tuning action must also be taken at run-time. At the time of submission of a job, the significant parts of it are identified and are instrumented in an appropriate manner. Significant parts (in this case loops) may be identified with the help of historical data or domain knowledge of the client. Instrumentation is done depending on the types of data the Analysis Agents require to collect. Upon the suggestion of Analysis Agent, the JobExecutionManager Agent invokes the Tuning Agent. The Tuning Agent receives the details of the performance problem, and initiates tuning action. During this process, the job remains suspended and resumes again only after the Tuning Agent sends a signal for resumption to the associated JobExecutionManager Agent.

An example will explain the preliminary implementation of the Tuning Agent. A job with a single significant loop (of depth 1) is considered for this purpose. As soon as the job begins, a ‘Ready’ message from the JobExecutionManager Agent activates the Analysis Agent. In turn, Analysis Agent sends a ‘Query’ message to the JobExecutionManager Agent with a specific value of the fraction of the total loop (e.g. 25%) that needs to be executed before collecting performance data. Then the job starts executing and when this specific percentage of the significant loop executes, the performance data related to the this execution is sent to the Analysis Agent and the job is suspended for the time being. All messages are transmitted in XML format. One such message format is shown in Figure 8.11. On the basis of the analysis of this performance data, Analysis Agent decides whether any tuning action to be taken for enhancement of the performance of the job. For example,

if the job initially starts executing on p (>=1) processors, the Analysis Agent may compute the estimated completion time (T_c) and find that the execution may complete within the expected completion time (T_{ect}) if more processors (>p) are provided at run-time. Thus, on the basis of the recommendation from the Analysis Agent, the remaining part (i.e. 75%) of the loop either continues its execution as before or the Tuning Agent tunes the remaining part of the job in order to run it on required number of processors.

```
<?xml version = "1.0">
<Execution_Strategy>
        <Total_iteration> 100000 </Total_iteration>
        <Exec_ iteration _percentage> 0.25 </Exec _iteration_ percentage >
        <Processor_used> 1 </Processor_used>
        <Execution_time_sec> 361.977 </Execution_time_sec>
</Execution_Strategy >
```

Figure 8.11: Reply message format.

Figure 8.12 shows a sequence diagram depicting the interactions among the JobExecutionManager Agent, Analysis Agent and Tuning Agent.

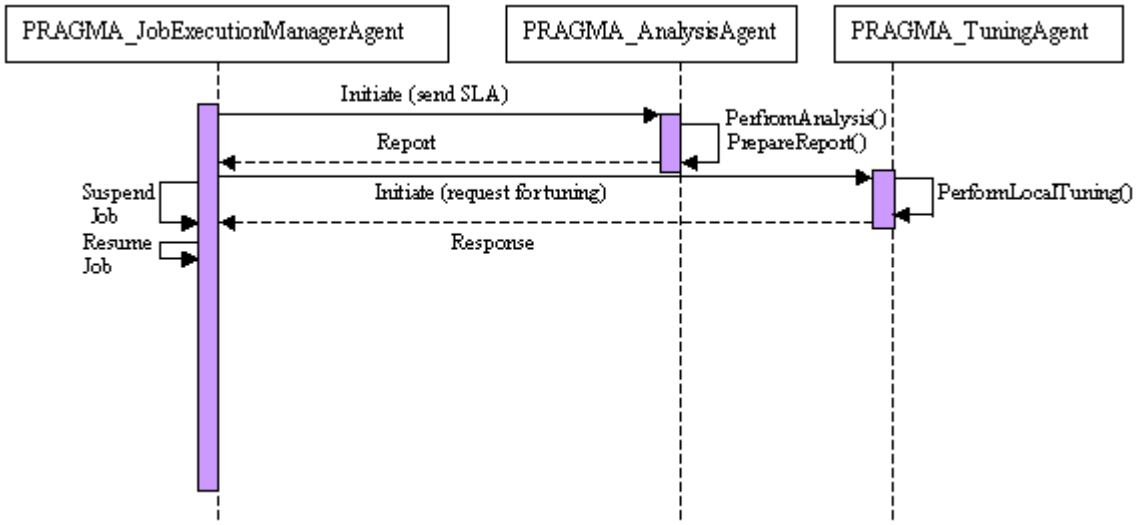


Figure 8.12: Sequence diagram for local tuning.

With the above strategy, the job actually runs in two different parts – the first part executes before making any enhancement and uses the resources as specified at the initial resource brokering phase. The second part executes after making the enhancement. Let T_{ne} be the execution time of the entire job if no enhancement is made, T_{ae} be the execution time of the entire job on enhanced resources, and N be the total number of instruction count of the job. Now, if N_p number of instructions of a job is run without enhancement, its execution time is given by $((T_{ne}/N)*N_p)$ and the execution time of the enhanced part is $((T_{ae}/N)*(N-N_p))$. Thus, the total execution time of the job T_e is given by,

$$T_e=((T_{ne}/N)*N_p))+O_e+((T_{ae}/N)*(N-N_p))$$

$$\text{Or, } T_e= T_{Np}+O_e+T_{at}$$

T_{Np} is the time taken by the job to complete N_p number of instructions on the particular resource provider (as mentioned in Section 7.5), i.e., $T_{Np}=((T_{ne}/N)*N_p)$.

T_{at} is the time spent by the job to execute the remaining part of its computation (after local tuning) on the same resource provider, i.e., $T_{at}=((T_{ae}/N)*(N-N_p))$.

O_e is the overhead for analysis and tuning activities.

Analysis Agent also analyzes the improvement in speedup and checks whether it achieves expected completion time of a job as mentioned in the SLA. The speedup is given by,

$$T_{ne}/T_e=T_{ne}/(T_{Np}+O_e+T_{at})$$

In this scenario, it must be checked that $T_{Np}+O_e+T_{at}\leq T_{ect}$.

Where, T_{ect} is the expected completion time of a job, which is mentioned in the SLA (see Section 7.5).

Thus, the Analysis Agent checks whether any real benefit can be envisaged after application of the tuning action. If it predicts any real benefit, then only the Tuning Agent is invoked.

8.6 Conclusion

This chapter mainly focuses on how adaptive execution features are implemented in PRAGMA. After submission of the jobs, if it is noted that the performance of a job is degrading or there is a violation of the SLA, PRAGMA takes necessary actions to improve the performance either locally with the help of a Tuning Agent or provide by dynamically changing the execution environment with the help of the JobController Agent, and the JobExecutionManager Agent. For this purpose, PRAGMA incorporates mobile agents.

Case studies of few existing mobile agent systems namely Aglet, Concordia, Naplet, Odyssey and Voyager are presented in this chapter. Implementation details of an agent framework using Jini have been explained in this chapter. A discussion has been presented concerning the performance model for the interaction of mobile agents within the PRAGMA. Implementations of the Analysis Agent and the Tuning Agent have not received much attention in this work. A preliminary implementation of the Analysis Agent has already been presented in Chapter 7. This chapter outlines the functioning of the Tuning Agent and its interaction with the Analysis Agent using a simple example of local tuning for performance enhancement.

The next chapter evaluates the adaptive execution ability in PRAGMA by using it for executing a batch of jobs (different test codes) on a local Grid test bed. The performance and efficacy of the Jini-based agent framework in PRAGMA is assessed by conducting a series of experiments and demonstrating the experimental results.

Chapter 9

Evaluation of Adaptive Execution by PRAGMA

The main objective of this chapter is to demonstrate the effectiveness of adaptive execution feature in PRAGMA. Theoretical and implementation details of this feature have been discussed in Chapter 8, which particularly concentrates on its two aspects - rescheduling and local tuning. Various techniques used in PRAGMA to improve the performance of jobs by adapting them to the dynamic availability and capacity of Grid resources have been described in Chapter 8. The current chapter presents the results of using these techniques in a Grid environment. A set of experiments has been carried out on a local Grid test bed. The chapter is organized as follows. The experimental set-up and the test codes are described in Section 9.1. The results of these experiments are depicted and discussed in Section 9.2 and Section 9.3. In particular, Section 9.2 demonstrates the benefits of rescheduling the jobs onto different resource providers and Section 9.3 illustrates the effectiveness of using local tuning techniques during the execution of the jobs.

9.1 Experimental Set-up

The major computational nodes in the local test bed used for the experimentation purposes are:

- Intel Pentium-4 PC of 2.8 GHz speed and 512 MB physical memory (referred as *server1*). Intel Pentium-4 PC has a single processor and running Linux kernel version 2.6.9-11.EL.

- HP NetServer LH 6000 of 700 MHz speed and 1024 MB physical memory (referred as *server2*). HP NetServer has two processors and running Linux kernel version 2.4.22-1.2115.npt1smp.
- IBM Power4 pSeries 690 (P-690) Regatta server of 1.9 GHz speed and 32 GB physical memory (referred as *server3*). Although, the P-690 server has 16 processors, for the purpose of this experimentation, one logical partition (among the four partitions) of the server with 7 processors and 14 GB memory has been used. The operating system used on this partition is SUSE Linux Enterprise Server9 kernel version 2.6.5-7.97-pSeries.

Test Codes

For experimentation purpose, SciMark 2.0 benchmark codes [96] for sequential C, sequential Java and parallel C applications and JGF (Java Grande Forum) benchmark codes [56] for parallel Java applications have been used. Minor modifications have been applied to the original source codes in order to run them in `PRAGMA` environment. Applications in C, Java, C OpenMP and Java OpenMP are executed to exhibit the interoperability of the mobile agent framework within `PRAGMA`. Mobile agent framework of `PRAGMA` is implemented using Java-based Jini and manages sequential C and parallel C (OpenMP) jobs using Java Native Interfaces (JNI). Jobs in parallel Java are implemented using Java OpenMP, i.e. Jomp [9]. Four different programs are considered which include:

- Sparse Matrix Multiplication (SciMark 2.0 and JGF benchmark): This program uses an unstructured sparse matrix stored in compressed-row format with a prescribed structure and exercises indirect addressing and non-regular memory references. A NxN sparse matrix with 5 times N nonzeros (NZ) is used. A set of experiments has been performed varying the size of the sparse matrix from 1000x1000 to 100000x100000. The main loop has been iterated for NZ number of times.
- LU matrix factorization (SciMark 2.0 and JGF benchmark): This program computes LU factorization followed by a triangular solve of a dense NxN matrix using partial pivoting and uses dense matrix operations. In the experiments performed with this

code, data sizes have been varied from 200x200 to 1000x1000. The main loop has been iterated for 20 times.

- Gaussian Elimination: This program provides solution for a system of Linear equations, Ax=B. There are two phases in the Gaussian Elimination Method. First phase is the forward elimination of variables x_1, x_2, x_3... x_n and the second phase performs backward substitution to find the values of the unknowns, x_n, x_{n-1}, x_{n-2} ... x_1. Data sizes for this code have been varied from 400 to 1000. The main loop has been iterated for 20 times.
- Matrix Multiplication: This program multiplies two NxN matrices. Matrix multiplication is chosen as a benchmark application because it is both data and compute-intensive. Data sizes in the experiments have been varied from 400x400 to 1000x1000. The main loop has been iterated for 100 times.

The next two sections present the results of the experiments carried out to demonstrate the efficiency of rescheduling and local tuning techniques used by PRAGMA.

9.2 Adaptive Execution by Rescheduling

In order to demonstrate the significance of rescheduling the jobs onto different resources, the following set of experiments has been carried out. Case 1 has been applied for sequential C and sequential Java codes. On the other hand, case 2 has been applied for parallel C and parallel Java codes. Each experiment has been conducted in various scenarios, which are compared in the results.

Case 1: Execution of jobs in sequential C and sequential Java.

*Scenario1***:** The job is initially scheduled to server2 and completes its execution on it.

*Scenario2***:** The job is initially scheduled to server2, and after completing a portion of its computation (in this case one fourth) is rescheduled to server3 (because the server2 either withdraws its services or its performance is not satisfactory as detected by the Analysis Agent) and executes the remaining part (rest three fourth) of the job on server3.

Case 2: Execution of jobs in parallel C and parallel Java.

*Scenario1***:** The job is initially scheduled to one resource provider and completes its execution there. The following experiments have been carried out in this scenario.

(a) The job is initially scheduled to server1 and completes its execution on it with a single processor.

(b) The job is initially scheduled to server2 and completes its execution on it with two processors.

*Scenario2***:** The job is initially scheduled to one resource provider, and after completing a part of its computation, a performance problem is indicated by the Analysis Agent on that resource provider and as per the recommendation of the Analysis Agent, the job is rescheduled to another resource provider. This is a specific case when the job requires more resources (here processors), but the current resource provider is unable to provide that. The following experiments have been carried out in this scenario.

(a) The job is initially scheduled to server1 (starts execution with one processor), and after completing a part (in this case half) of its computation is rescheduled to server2 and executes the remaining part of the job on server2 on 2 processors.

(b) The job is initially scheduled to server2 (starts execution on two processors), and after completing a part (36.5%) of its computation is rescheduled to server3 and executes the remaining part of the job on server3 with 4 processors.

(c) The job is initially scheduled to server2 (starts execution on two processor), and after completing a part (36.5%) of its computation is rescheduled to server3 and executes the remaining part of the job on server3 with 6 processors.

In all the experiments of Scenario2, overheads are involved while migrating the job from one server to another server. These overheads are due to additional time required for discovering the services, rescheduling the jobs and establishing SLAs on a second resource provider.

Case 1: Sequential C Code

The sparse matrix multiplication benchmark code in sequential C has been used for this purpose. Table 9.1 compares the two scenarios to demonstrate the effect of rescheduling. Figure 9.1 and Figure 9.2 depict the results and compare the total time taken by the JobExecutionManager Agent for sparse matrix multiplication job. Execution times are measured with varying data sizes.

Sparse Matrix Multiplication (NZ number of iterations) **Data Size**	Scenario1 Total time on Server2 (Sec)	**Scenario2** Rescheduling Total time (Sec)	Time spent on Server2 before migration (Sec)	Time spent on Server3 after migration (Sec)	Overhead (Sec)
N=1000,NZ=5000	0.907	1.129	0.256	0.345	0.528
N=2000,NZ=10000	3.644	2.926	0.922	1.273	0.731
N=4000,NZ=20000	14.551	9.916	3.601	4.926	1.389
N=8000,NZ=40000	84.280	38.837	17.728	19.128	1.981
N=16000,NZ=80000	509.91	170.005	124.136	33.259	2.61

Table 9.1: Sparse matrix multiplication - execution performance of job, initially scheduled to Server2 and after one fourth computation migrated to Server3, times are in seconds.

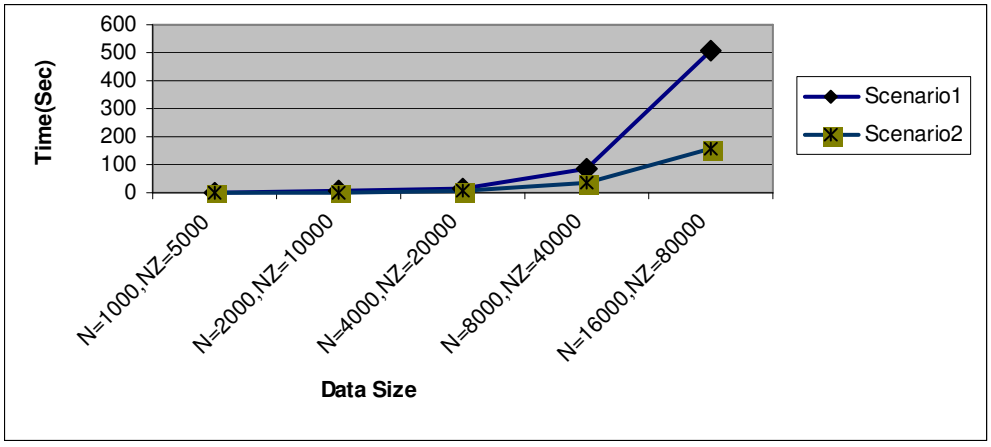


Figure 9.1: Sparse matrix multiplication - comparison of performances in Scenario1 and Scenario2.

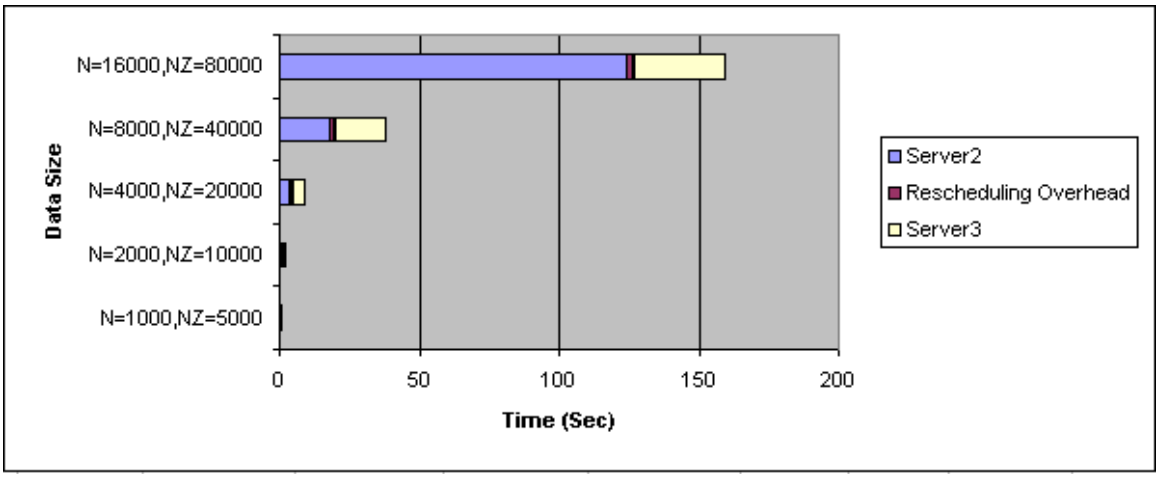


Figure 9.2: Sparse matrix multiplication - results of rescheduling from Server2 to Server3 in case of Scenario2.

Case 1: Sequential Java Code

In this experiment, LU factorization benchmark code and sparse matrix multiplication benchmark code in Java have been used. The total time required to complete the execution of LU factorization in both scenarios as well as the time spent on each server in the second scenario and the overhead for rescheduling are shown in Table 9.2. Table 9.3 presents the same measurements for sparse matrix multiplication. Figure 9.3 and 9.4 compare the total time taken by the JobExecutionManager Agent for LU factorization job. Figure 9.5 and 9.6 shows the same for sparse matrix multiplication.

LU Factorization (20 iterations) **Data Size**	Scenario1 Total time on Server2 (Sec)	**Scenario2** Rescheduling Total time (Sec)	Time spent on Server2 before migration (Sec)	Time spent on Server3 after migration (Sec)	Overhead (Sec)
200x200	2.116	4.101	0.359	1.528	2.214
400x400	14.3	15.668	4.212	3.646	7.81
600x600	73.932	53.269	20.502	11.826	20.941
800x800	194.669	113.509	60.429	17.226	35.854
1000x1000	432.328	250.093	117.812	73.132	59.149

Table 9.2: LU factorization - execution performance of job, initially scheduled to Server2 and after one-fourth computation migrated to Server3, times are in seconds.

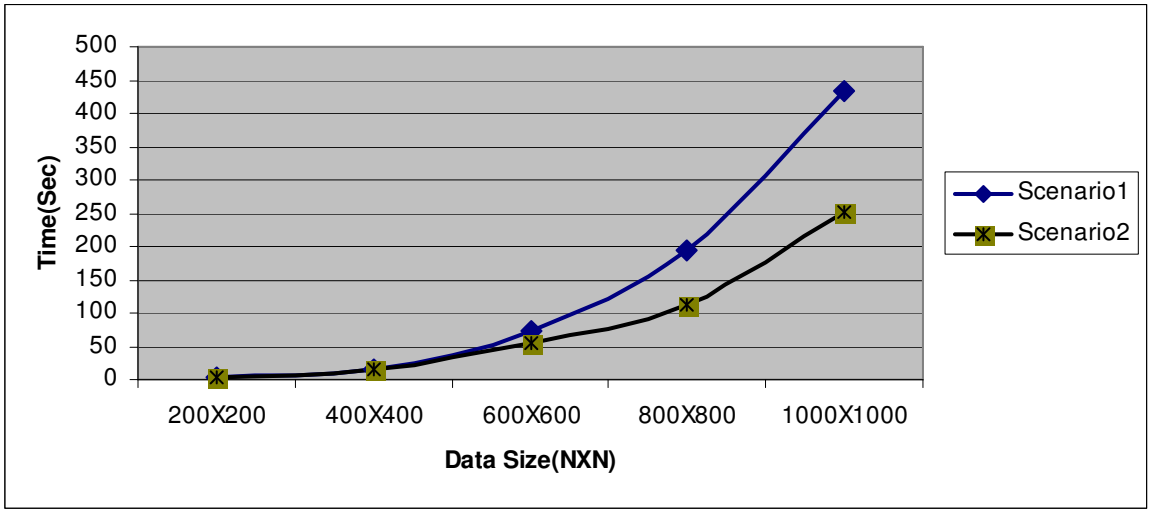


Figure 9.3: LU factorization - comparison of performances in Scenario1 and Scenario2.

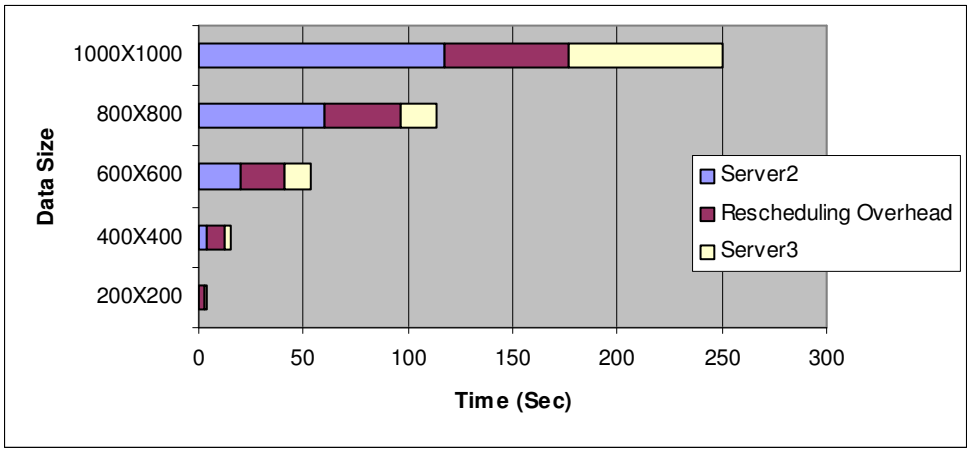


Figure 9.4: LU factorization - results of rescheduling from Server2 to Server3 in case of Scenario2.

Sparse Matrix Multiplication (NZ number of iterations) **Data Size**	Scenario1 Total time on Server2 (Sec)	**Scenario2** Rescheduling Total time (Sec)	Time spent on Server2 before migration (Sec)	Time spent on Server3 after migration (Sec)	Overhead (Sec)
N=1000,NZ=5000	1.019	1.104	0.219	0.855	0.03
N=2000,NZ=10000	4.527	2.824	0.842	1.806	0.176
N=4000,NZ=20000	18.149	9.112	3.529	5.207	0.376
N=8000,NZ=40000	77.761	37.208	16.258	20.518	0.432
N=16000,NZ=80000	498.908	166.872	127.389	37.633	1.85

Table 9.3: Sparse matrix multiplication - execution performance of job, initially scheduled to Server2 and after one fourth computation migrated to Server3, times are in seconds.

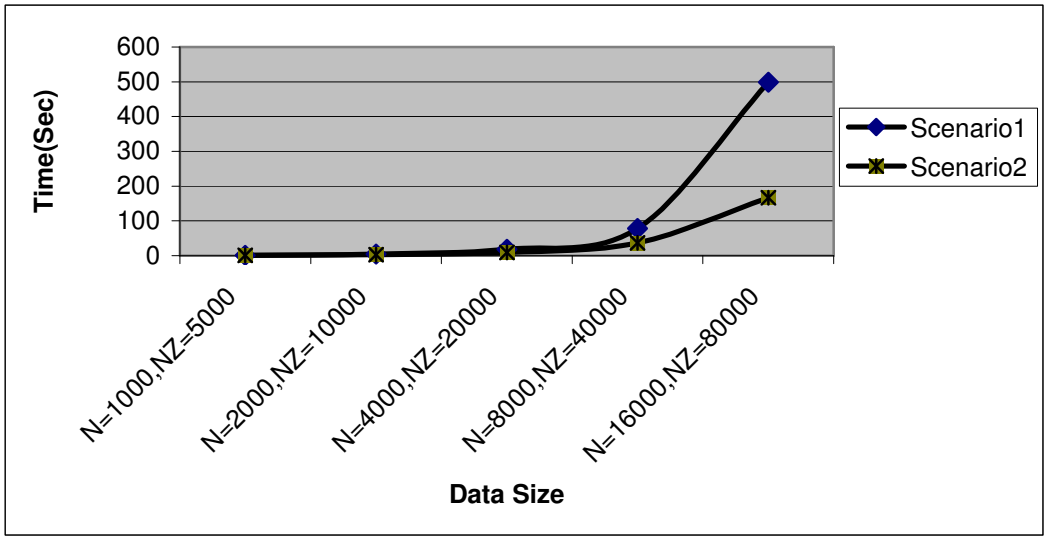


Figure 9.5: Sparse matrix multiplication - comparison of performances in Scenario1 and Scenario2.

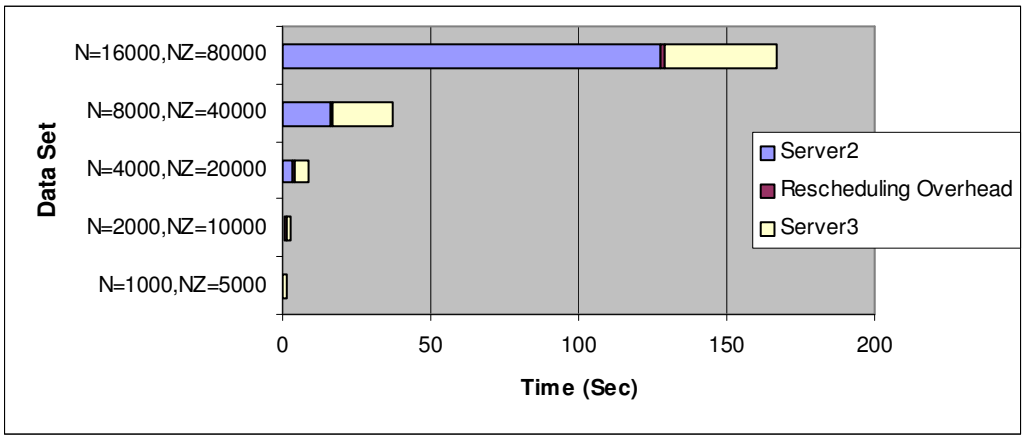


Figure 9.6: Sparse matrix multiplication - results of rescheduling from Server2 to Server3 in case of Scenario2.

Discussion

As mentioned in Section 7.5 and Section 8.1, the Analysis Agent of PRAGMA recommends to reschedule a job if it finds that the estimated completion time of the job is greater than the expected completion time mentioned in the SLA and the current resource provider is either overloaded or unable to fulfill its promise (as is expressed when it accepts the SLA). However, a major overhead involved with this process is the overhead due to migration and controlling the agents. The results show that even for data sets of moderate sizes the associated overheads do not significantly affect the total execution time. In case of sparse matrix multiplication, this overhead is less compared to LU factorization because of the data sizes. An NxN sparse matrix with NZ non-zeros is stored in a compressed row format, which reduces the size of the used data set. Compressed Row Storage (CRS) puts the subsequent non-zeros of the matrix rows in contiguous memory locations. It needs three vectors: one of size NZ to hold the non-zero matrix components; and the other two integer arrays, one of size NZ to hold the column indices of the non-zero matrix components and the other of size N+1 to store the location in vector of the start of each new row. Thus, a total of (2xNZ+(N+1)) size of storage space is required which is much less than the NxN size. On the other hand, the LU factorization stores in NxN matrix format. The computation involves serialization, transfer and deserialization of the entire data set. Although, it is clear

that rescheduling is good choice in both cases, the size of data to be moved is important (as expected). Thus, besides monitoring the performance of the job, the Analysis Agent should also use the performance model described in Section 8.4 to decide whether to migrate the job.

In the above set of experiments, significant improvement in running time could be achieved due to the difference in CPU clock speeds of server2 and server3. The experiments demonstrate that rescheduling can be done successfully through mobile agents using the Jini-based framework. This particular set of experiments also highlights that for small computational jobs migration might not be a right decision. However, for large computations, it can possibly be a good decision. The results show that in spite of an increase in migration overhead, the advantages due to migration may still compensate the overhead and disadvantages of executing the jobs on different servers. Thus, the decision of migration becomes effective if the following situation occurs in Case 1:

Let T_e be the total execution time when the job is executed completely on server S1. In scenario2, let T_{S1} be the time spent on server S1 before migration, T_{S2} be the time spent on server S2 and T_m be the total overhead involved with migration of the job from server S1 to server S2. Migration is beneficial only if the Analysis Agent estimates that $T_e > T_{S1} + T_m + T_{S2}$.

The condition, $T_{Np} + O_e + T_{at} \leq T_{ect}$, as discussed in Section 8.5 must also be satisfied. Here, T_{Np}, O_e and T_{at} are equal to T_{S1}, T_m and T_{S2} respectively.

Case 2: Parallel C Code

To demonstrate the effectiveness of the mobile agent framework for parallel C codes, the gauss elimination code has been used. Table 9.4 depicts the effect of rescheduling of parallel jobs by comparing the outcomes of the above mentioned Scenario1(a) and Scenario2(a) of Case 2.

Gauss Elimination **(20 iterations) Data Size (number of unknowns)**	Scenario1(a) Total time on Server1 (With one Processor) (Sec)	**Scenario2(a)** Rescheduling Total time (Sec)	Time spent on Server1 before migration (With one Processor) (Sec)	Time spent on Server2 after migration (With two Processors) (Sec)	Overhead (Sec)
400	33.047	32.671	16.602	7.689	8.38
600	124.245	108.116	64.856	25.995	17.265
800	317.412	265.291	166.374	68.092	30.825
1000	637.203	537.503	341.618	142.985	52.9

Table 9.4: Gauss Elimination - results of rescheduling from Server1 to Server2.

It is clearly evident from the data presented in the table, that using two processors on the Server2, performance of the code is improved. Figure 9.7 and Figure 9.8 depict the results and compare the total time taken to execute the gauss elimination job in two scenarios of Case 2. Execution times are measured with varying numbers of unknowns.

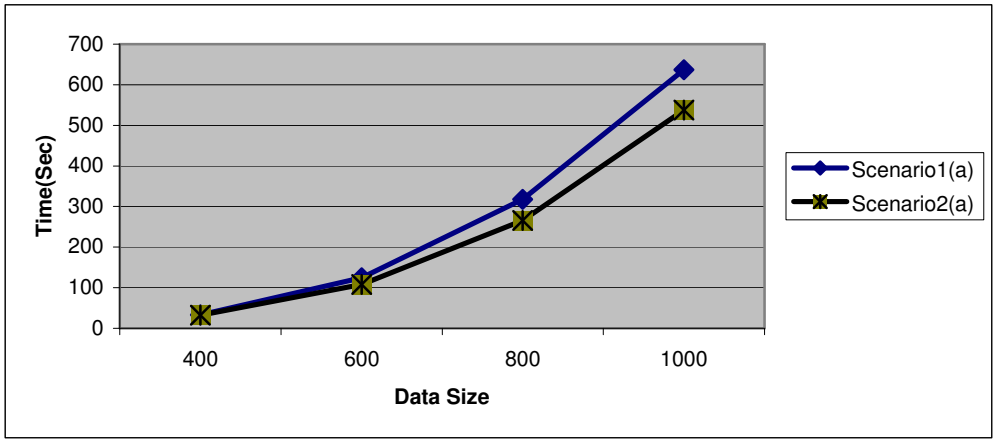


Figure 9.7: Gauss Elimination - comparison of performances in Scenario1(a) and Scenario2(a).

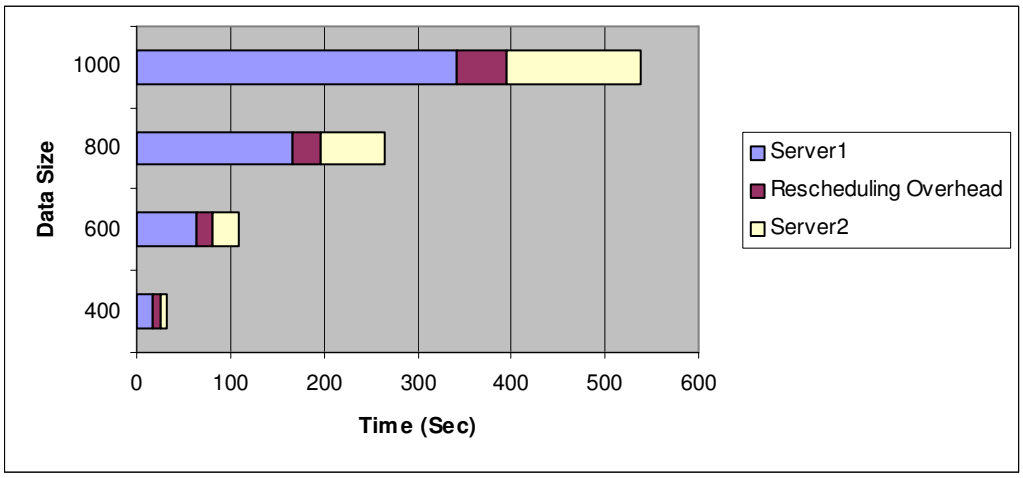


Figure 9.8: Gauss Elimination - results of rescheduling from Server1 to Server2 in case of Scenario2(a).

Case 2: Parallel Java Code

Sparse matrix multiplication code in parallel Java has been used to demonstrate the effectiveness of the framework for parallel Java. Table 9.5 depicts the effect of rescheduling of parallel job by comparing the outcomes of the previously mentioned Scenario1(b) and Scenario2(b) of Case 2. Table 9.6 shows the results of a similar experiment, but in this case, the number of processors used on the second server is six. Results from Scenario1(b) and Scenario2(c) of Case 2 are compared in the second experiment.

Figure 9.9 and Figure 9.10 depict the results and compare the total time taken by the JobExecutionManager Agent in Scenario1(b) and Scenario2(b) of Case 2 for sparse matrix multiplication job.

Sparse Matrix Multiplication **(NZ number of iterations) Data Size**	Scenario1(b) Total time on Server2 (With two Processor) (min)	**Scenario2(b)** Rescheduling Total time (min)	Time spent on Server2 before migration (With two Processor) (min)	Time spent on Server3 after migration (With four Processor) (min)	Overhead (min)
N=10000,NZ=50000	4.845	2.052	1.768	0.094	0.188
N=20000,NZ=100000	15.640	6.241	5.708	0.327	0.204
N=30000,NZ=150000	30.565	12.137	11.15645	0.746	0.234
N=40000,NZ=200000	56.741	22.387	20.710	1.352	0.323
N=50000,NZ=250000	91.973	36.307	33.570	2.313	0.423

Table 9.5: Sparse matrix multiplication - execution performance of job, initially scheduled to Server2 (with two processors) and after 36.5% of the entire computation migrated to Server3 (with four processors), times are in minutes.

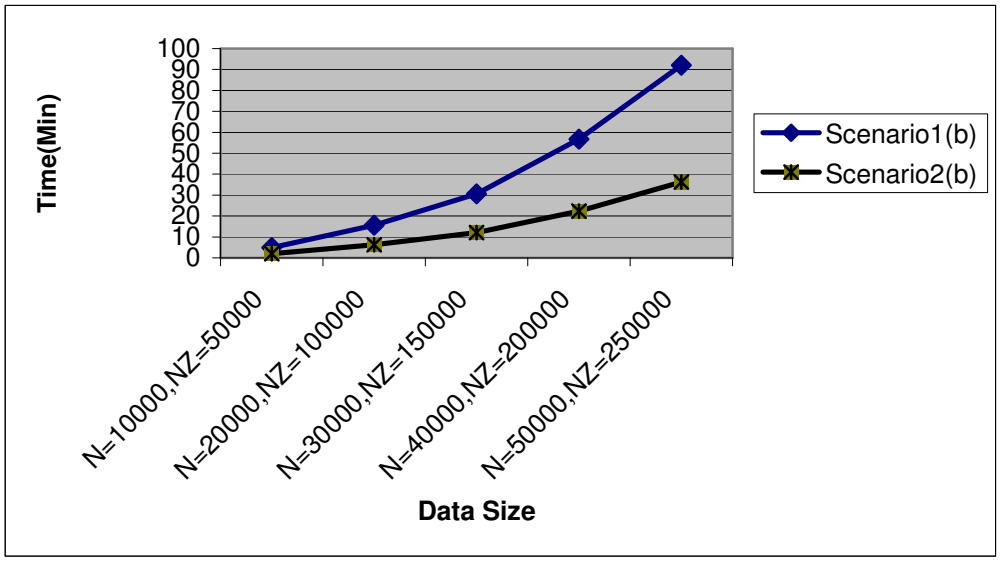


Figure 9.9: Sparse matrix multiplication - comparison of performances in Scenario1(b) and Scenario2(b).

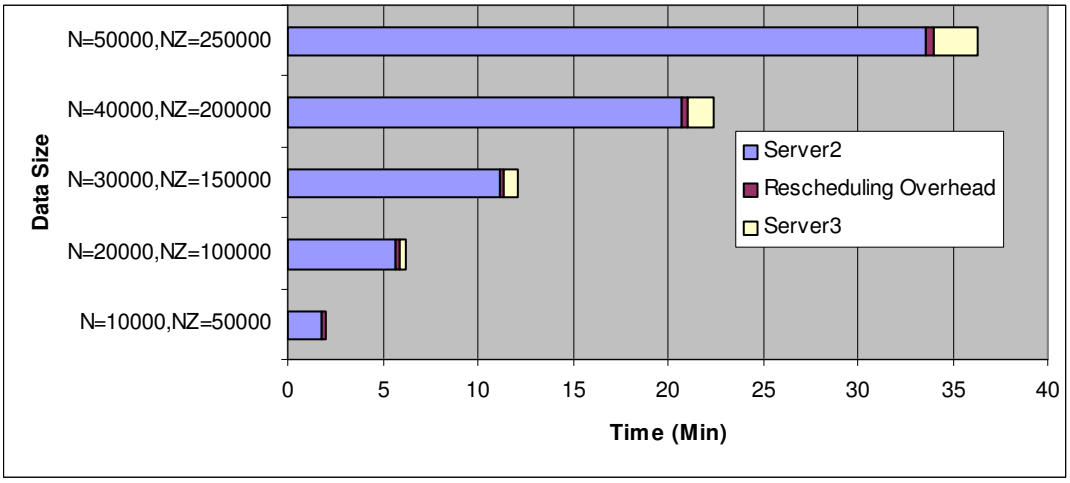


Figure 9.10: Sparse matrix multiplication - results of rescheduling from Server2 to Server3 in case of Scenario2(b).

Figure 9.11 and Figure 9.12 depict the results and compare the total time taken by the JobExecutionManager Agent in Scenario1(b) and Scenario2(c) of Case 2 for sparse matrix multiplication job.

Sparse Matrix Multiplication **(NZ number of iterations)** **Data Size**	Scenario1(b) Total time on Server2 (With two Processor) (min)	**Scenario2(c)** Rescheduling Total time (min)	Time spent on Server2 before migration (With two Processor) (min)	Time spent on Server3 after migration (With six Processor) (min)	Overhead (min)
N=10000,NZ=50000	4.797	2.018	1.751	0.071	0.195
N=20000,NZ=100000	15.629	6.155	5.704	0.250	0.200
N=30000,NZ=150000	30.734	11.993	11.218	0.514	0.260
N=40000,NZ=200000	54.856	21.314	20.022	0.909	0.382
N=50000,NZ=250000	84.058	32.551	30.681	1.432	0.437

Table 9.6: Sparse matrix multiplication - execution performance of job, initially scheduled to Server2 (with two processors) and after 36.5% of the entire computation migrated to Server3 (with six processors), times are in minutes.

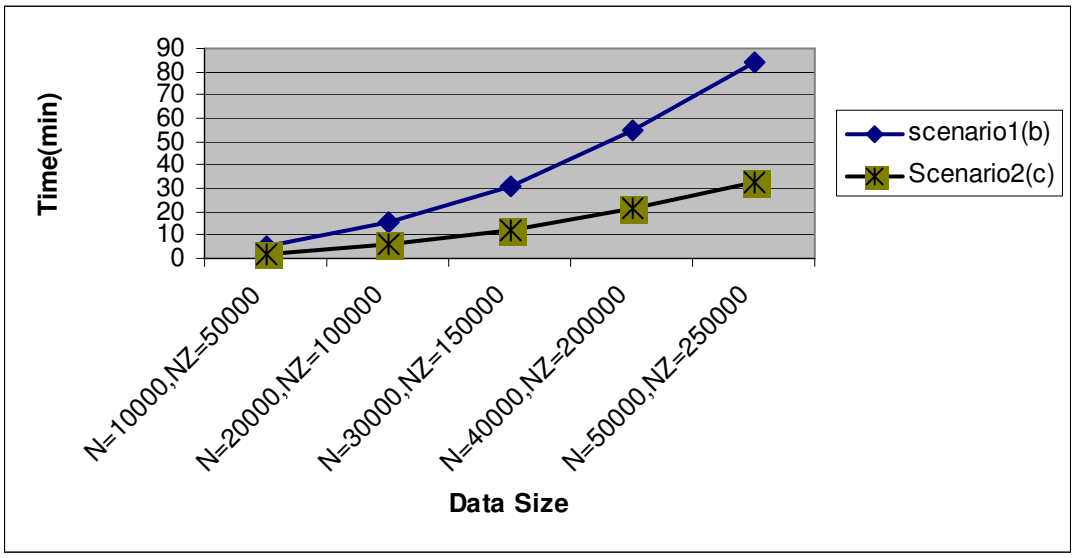


Figure 9.11: Sparse matrix multiplication - comparison of performances in Scenario1(b) and Scenario2(c).

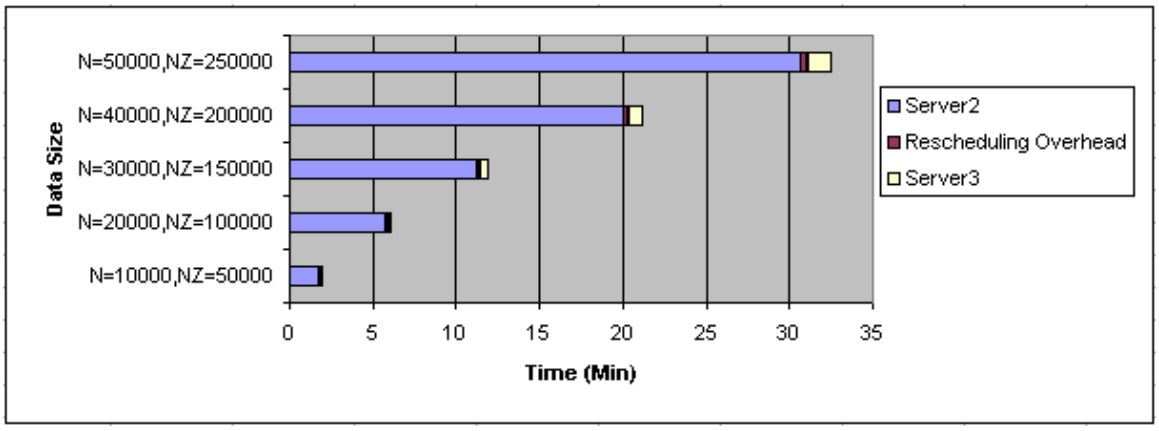


Figure 9.12: Sparse matrix multiplication - results of rescheduling from Server2 to Server3 in case of Scenario2(c).

Discussion

Above results clearly indicate that migration is a good decision when $T_e > T_{S1} + T_m + T_{S2}$ condition is satisfied, i.e. when the term T_m (associated overhead) becomes insignificant compared to the benefit achieved because of migration. This experiment demonstrates the cases when an adaptive execution environment can be offered by PRAGMA. Experiment in Case 2 for parallel C code has been performed to compare the results from Scenario1(a)

and Scenario2(a). Experiment in Case 2 for parallel Java code has been carried out with two arrangements: (i) to compare the results from Scenario1(b) and Scenario2(b), and (ii) to compare the results from Scenario1(b) and Scenario2(c). In both experiments, the decision of rescheduling has been proved to be beneficial. For obvious reason, when the job migrates to Server3 and executes with six processors, outcome of the experiment shows the best result (see Figure 9.12). Rescheduling overhead is involved in all these experiments, but in respect to the performance improvement of the suffered job it is negligible.

9.3 Adaptive Execution by Local Tuning

Although rescheduling is an important aspect of adaptive execution and receives more focus in this thesis, a job must first be attempted to be tuned locally to improve the performance of the job and to maintain the SLA between the client and the resource provider. The thesis implements a simple approach to local tuning as has been described in Section 8.5. This section presents the results of the implementation.

The experiments have been carried out in the situations when an Analysis Agent has suggested for local tuning as the performance of the job could be enhanced using additional resources. The set-up discussed in Section 9.1 has been used for this set of experiments as well. Two scenarios have been used for comparing the results of these experiments.

*Scenario1***:** The job is scheduled to server3 and completes its execution on it with a single processor.

Scenario2: The job is scheduled to server3 starts executing using a single processor on a multi-processor machine. After completing a part of its computation, `PRAGMA` increases number of threads at run-time (as per the recommendation from the Analysis Agent) and executes the remaining part of the job on that server with:

(a) Two processors

(b) Four processors

(c) Six processors

Parallel C Job

In this section, the results for tuning a matrix multiplication code with OpenMP directives will be presented. Table 9.7, Table 9.8 and Table 9.9 show the effects of local tuning by comparing the results of Scenario1 with Scenario2(a), Scenario2(b) and Scenario2(c) respectively for different sizes of the matrices. Figure 9.13 depicts the results of comparison.

Matrix Multiplication **(100 iterations)** **Data Size**	Scenario1 Total time on Server3 (With one processor) (Sec)	**Scenario2(a)** Time spent on Server3 (With one processor) (Sec)	**Scenario2(a)** Time spent on Server3 (With two processors) (Sec)	**Performance Improvement (%)**
400x400	34.524	8.631	12.95	37.5
600x600	188.516	47.129	66.888	39.5
800x800	606.58	151.645	217.986	39.1
1000x1000	1447.908	361.977	516.227	39.3

Table 9.7: Matrix multiplication – executes on Server3 with one processor and after completing one fourth of the entire computation, executes the remaining part on the same server with two processors, times are in seconds.

Matrix Multiplication **(100 iterations)** **Data Size**	Scenario1 Total time on Server3 (With one processors) (Sec)	**Scenario2(b)** Time spent on Server3 (With one processor) (Sec)	**Scenario2(b)** Time spent on Server3 (With four processors) (Sec)	**Performance Improvement (%)**
400x400	35.304	8.826	6.692	56.6
600x600	192.084	48.021	34.664	57.4
800x800	610.504	152.626	103.211	58.3
1000x1000	1447.312	361.828	240.858	58.3

Table 9.8: Matrix multiplication - executes on Server3 with one processor and after completing one fourth of the entire computation, executes the remaining part on the same server with four processors, times are in seconds.

Matrix Multiplication **(100 iterations)** **Data Size**	Scenario1 Total time on Server3 (With one processor) (Sec)	**Scenario2(c)** Time spent on Server3 (With one processor) (Sec)	**Scenario2(c)** Time spent on Server3 (With six processors) (Sec)	**Performance Improvement (%)**
400x400	33.712	8.428	4.898	59.9
600x600	188.904	47.226	23.511	62.6
800x800	604.512	151.128	67.837	63.7
1000x1000	1447.536	361.884	161.34	63.8

Table 9.9: Matrix multiplication - executes on Server3 with one processor and after completing one fourth of the entire computation, executes the remaining part on the same server with six processors, times are in seconds.

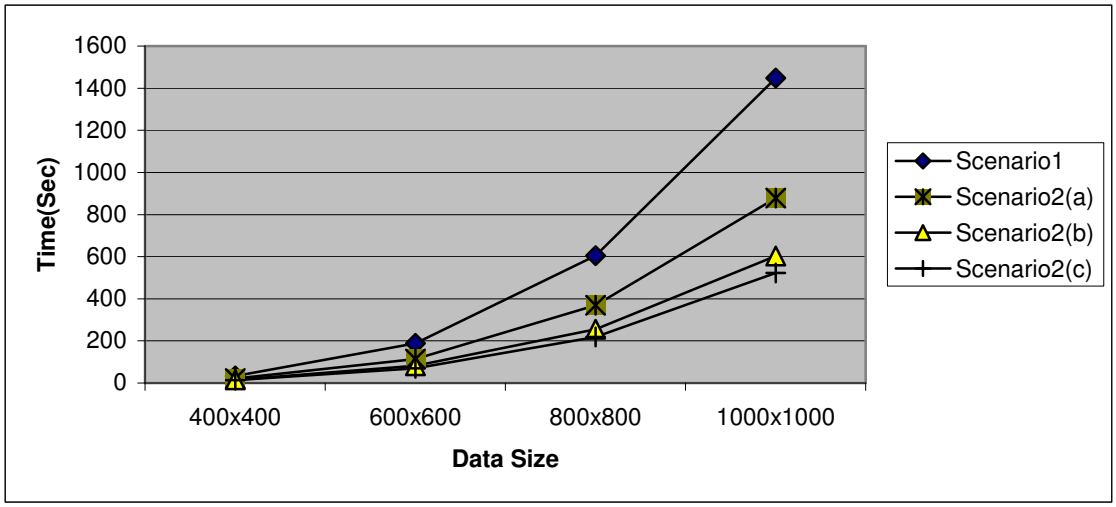


Figure 9.13: Matrix multiplication - comparison of performances in Scenario1 and Scenario2(a), Scenario2(b), Scenario2(c).

Parallel Java Job

In this section, results of tuning a sparse matrix multiplication code with Jomp directives will be presented. Table 9.10, Table 9.11 and Table 9.12 show the effect of local tuning to improve the performance of this job by comparing the results of Scenario1 with Scenario2(a), Scenario2(b) and Scenario2(c). Figure 9.14 depicts the results of this comparison.

Sparse Matrix Multiplication **(NZ number of iterations)** **Data Size**	Scenario1 Total time on Server3 (With one processor) **(min)**	**Scenario2(a)** Time spent on Server3 (With one processor) (min)	**Scenario2(a)** Time spent on Server3 (With two processors) (min)	**Performance Improvement (%)**
N=25000,NZ=125000	4.879	1.219	1.009	54.3
N=50000,NZ=250000	19.466	4.866	4.204	53.4
N=75000,NZ=375000	44.168	11.042	9.433	53.6
N=100000,NZ=500000	78.431	19.607	16.978	53.4

Table 9.10: Sparse matrix multiplication – executes on Server3 with one processor and after completing 36.5% of the entire computation, executes the remaining part on the same server with two processors, times are in minutes.

Sparse Matrix Multiplication **(NZ number of iterations)** **Data Size**	Scenario1 Total time on Server3 (With one processor) **(min)**	**Scenario2(b)** Time spent on Server3 (With one processor) (min)	**Scenario2(b)** Time spent on Server3 (With four processors) (min)	**Performance Improvement (%)**
N=25000,NZ=125000	4.780	1.195	0.528	64
N=50000,NZ=250000	19.322	4.830	2.070	64.2
N=75000,NZ=375000	43.888	10.972	4.689	64.3
N=100000,NZ=500000	78.711	19.677	8.455	64.3

Table 9.11: Sparse matrix multiplication – executes on Server3 with one processor and after completing 36.5% of the entire computation, executes the remaining part on the same server with four processors, times are in minutes.

Sparse Matrix Multiplication **(NZ number of iterations)** **Data Size**	Scenario1 Total time on Server3 (With one processor) **(min)**	**Scenario2(c)** Time spent on Server3 (With one processor) (min)	**Scenario2(c)** Time spent on Server3 (With six processors) (min)	**Performance Improvement (%)**
N=25000,NZ=125000	4.877	1.219	0.350	67.8
N=50000,NZ=250000	19.530	4.882	1.420	67.7
N=75000,NZ=375000	43.979	10.994	3.160	67.8
N=100000,NZ=500000	79.857	19.964	5.796	67.7

Table 9.12: Sparse matrix multiplication – executes on Server3 with one processor and after completing 36.5% of the entire computation, executes the remaining part on the same server with six processors, times are in minutes.

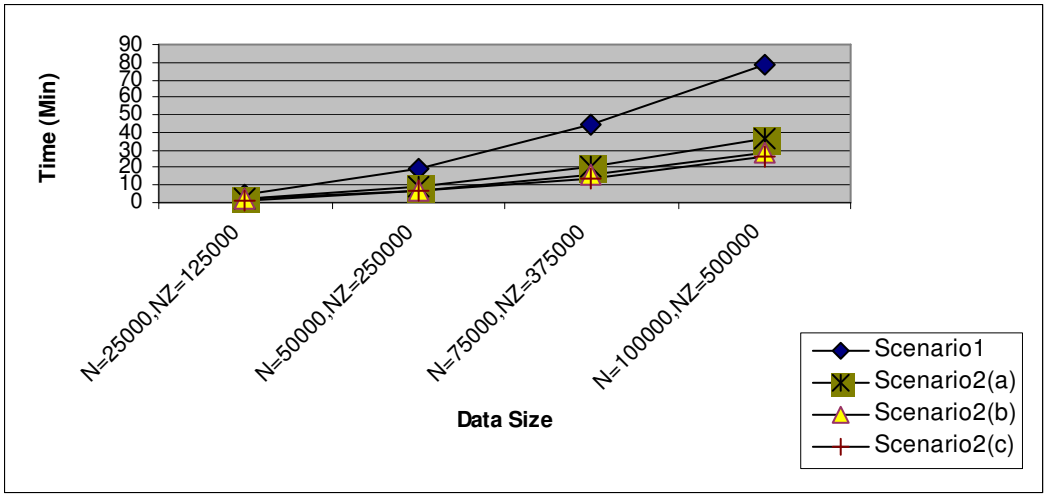


Figure 9.14: Sparse matrix multiplication - comparison of performances in Scenario1 and Scenario2(a), Scenario2(b), Scenario2(c).

Discussion

The results presented in this section provide adequate evidence that PRAGMA is capable of improving performance of a job to maintain QoS. In both experiments performance improves (as expected) by increasing number of threads. This experiment specifically portrays a special situation when the performance of a job can be improved by providing additional local resources (in this case processor). In this situation performance is much better as rescheduling is not required and thus migration overhead can be avoided. The experimental results also satisfy the condition $T_{Np}+O_e+T_{at} \leq T_{ect}$ (see Section 8.5). Here, T_{Np} is the time spent on server3 with one processor and T_{at} is the time spent on same node i.e., server3 with enhanced resources and O_e is the overhead involved in analysis and tuning (also include the overheads for controlling the agents), as discussed in Section 8.5.

9.4 Conclusion

The results presented in this chapter indicate that the decisions regarding rescheduling and local tuning on the basis of runtime analysis can generally help to achieve desired high-performance. The experimental results also demonstrate that the overhead involved here is nominal in comparison to the achieved speedup due to improvement in execution environment. The set of experiments gives some idea about the benefits of using PRAGMA,

although further experiments need to be carried out involving complex environments and complex test codes.

In a larger Grid environment with resources administered by multiple organizations, the task of runtime analysis plays an important role. The analyzer should be able to measure the performance correctly and estimate the completion time on the basis of the performance results and take appropriate decisions in real time. Thus, proper design of an analyzer is crucial for efficient implementation of PRAGMA tool. The concluding chapter of this thesis presents a hierarchical design of an analysis agent framework and future research may focus on the implementation of this framework.

Chapter 10

Conclusion

Grid computing has been gaining importance fast due to the ability to deal with continuously growing computational demands. However, to enable the efficient use of available resources further enhancements of the existing resource management systems are required. This research work is dedicated to the development of efficient resource management system. The thesis concludes with a review of the work presented in this thesis. The significance of the major results is summarised. A brief survey of related work is presented. Outstanding issues are discussed and directions for future work are suggested.

10.1 Overview of Thesis

This thesis has been motivated by the problem of managing multiple concurrent jobs executing in Grid for achieving high performance. The goal of this work is to design and implement a multi-agent framework, which provides fair resource allocation for complex Grid applications (decomposed into multiple jobs) with objective of overall cost optimization for resource usage. The framework also provides support to adaptive execution if job performance degrades because of changes in the Grid resources.

The significant contribution of this work is an integrated framework for performance based resource management for a batch of multiple concurrent jobs. The framework consists of four components: a Resource Broker, a Job Controller, an Analyzer and a Performance

Tuner (see Chapter 3). Gaia methodology has been used to design a multi-agent system (MAS) on the basis of this framework (as discussed in Chapter 4). The design of the multi-agent system has been presented using a number of models as proposed in Gaia methodology. These include role model and interaction model in the architectural design phase and agent model and services model in the detailed design phase. Altogether six agents have been identified to work within this framework; these are Broker Agent, ResourceProvider Agent, JobController Agent, JobExecutionManager Agent, Analysis Agent and Tuning Agent. Each agent has a distinct set of responsibilities and plays a specific role within the framework. The multi-agent system is designed to provide services for resource brokering, regular monitoring of jobs and the environment, and adaptive execution.

Resource Broker plays a crucial role in this framework. The aim of the resource broker is to find, select, reserve and allocate suitable resources to each job submitted by the client. A resource broker always tries to optimize resource utilization and ensure fairness. The framework uses a hybrid approach to resource brokering. Resource brokering at the time of initial scheduling of a batch of jobs is centralized, whereas resource brokering for rescheduling purpose is performed in distributed manner. This thesis also provides an overall cost optimization technique for executing a batch of multiple concurrent jobs (as discussed in Chapter 5) and proves its effectiveness. The resource selection algorithm presented in this thesis gives cost optimal solution for initial scheduling of jobs. Time complexity of this algorithm is $O(N^3)$, where NxN is the size of the cost matrix.

The framework also ensures that all jobs in a batch get the quality of service, which the client has desired and the involved resource providers have promised to offer. Regular monitoring of the execution of jobs and the infrastructure is an important feature of this framework and this helps to maintain the promised QoS and to achieve desired high performance (see chapter 7). Adaptive execution facility also improves job performance by making it accustomed to the dynamic availability and capacity of Grid resources. If job performance degrades, necessary actions are taken in order to maintain the promised QoS.

These actions include local tuning and rescheduling. The concept of mobile agent is used for initial scheduling and rescheduling of jobs. This thesis describes how the performance of a suffered job can be improved by adapting it to the dynamic availability of high-end Grid resources (as discussed in Chapter 8).

This thesis also demonstrates the implementation of the multi-agent system as a tool called PRAGMA and proves its efficiency in meeting the requirements of Grid applications (see Chapter 6 and Chapter 9). All the required functionalities of the multi-agent system are incorporated in PRAGMA.

10.2 Significance of Major Results

The major results of the thesis are

- A systematic and principled design of the MAS using Gaia methodology for performance based resource management in computational Grid (as discussed in Chapter 4) that enabled correct and successful implementation of PRAGMA. Different phases of design using Gaia methodology help to identify distinct features of MAS and output a concrete and fruitful design for PRAGMA.
- An evaluation of PRAGMA regarding its ability for overall cost / time optimized resource usage for a batch of jobs (as illustrated in Chapter 6). Resource brokering for multiple concurrent jobs to achieve overall cost / time optimization has not been particularly handled by the existing resource brokers. This multi-agent system provides suitable resource providers for a batch of jobs as well as tries to optimize the resource usage cost for the entire batch (not a single job), and as a result the entire batch of jobs is benefited.
- An evaluation of PRAGMA regarding its ability for adaptive execution of jobs to achieve desired QoS (as illustrated in Chapter 9). Adaptive execution by rescheduling and local tuning improves the performance of a suffered job. This procedure is complex due to the dynamic nature of the environment. However, the multi-agent system presented in this thesis evolves efficient techniques to reduce the overheads involved in dynamic adaptability of jobs.

Thus, this thesis evaluates (i) overall cost optimized resource usage and (ii) adaptive execution by PRAGMA for multiple concurrent jobs. The evaluations have been done with the help of two experimental studies, the results of which are presented in Chapter 6 and Chapter 9, respectively. The results presented in Chapter 6 demonstrate the effectiveness of initial resource selection algorithm based on Hungarian method in comparison with resource selection method based on min-min heuristic. These results demonstrate that the initial resource selection algorithm based on Hungarian method is a successful attempt for overall cost optimized usage of resources for a batch of jobs.

Adaptive execution of suffered jobs to maintain QoS is efficiently managed by PRAGMA. Chapter 9 demonstrates the result in evidence of this claim. The first set of results in Section 9.2 shows that mobile agent-based rescheduling can be done efficiently and performance of the suffered job can be enhanced. On the other hand, the second set of results in Section 9.3 provides evidence in support of the claim that local tuning techniques can be efficiently employed to improve the performance of a job. These results demonstrate that in spite of an increase in overhead due to managing the activities like rescheduling or local tuning, the techniques can be used to improve the performance of the suffered job.

10.3 Related Work

Current research in Grid computing is gradually focusing on the provision of an integrated environment for performance based Grid resource management systems. A number of projects are investigating this field. The ICENI project puts emphasis on the development of a component framework for generality and simplicity of use in Grid environment [37]. ICENI provides a platform independent framework, which is built using Java and Jini. ICENI tools provide optimized deployment of Grid application components via performance guided implementation selection. Component based application model is used in ICENI, where domain specific knowledge is encapsulated within software components. Complete applications are defined as a collection of one or more of these components, which helps scientists to combine their expertise. ICENI also supports dynamic extension by its ability to instantiate and connect new components to an existing deployed

application. Multiple steering and visualization components can also be configured to provide collaborative interaction sessions between trusted members. Minimum execution time and minimum execution cost are two constraints of user requirements, which are implemented within ICENI.

In order to address the issues related to the dynamicity in Grid environment, GrADS project [44] is working for integrating application monitoring and adaptive control services within their framework. The project aims at building a closed-loop control system on this framework which uses dynamic performance information and guides an application to completion despite performance variations in the underlying resource base via a process of adaptive control of both application behavior and resource demands. An application manager is associated with each application for monitoring the performance of that application for QoS achievement. Performance contracts are used to formalize the relationship between application requirements and resource capabilities. Rescheduling or redistribution of resources occurs when QoS contracts are not maintained. GrADS employs the application specific execution model and follows the AppLeS [6] approach for scheduling. GrADS monitor resources using NWS and use Autopilot for performance prediction [92, 115]. VGrADS [109], which extends the GrADS project, is also exploring performance tuning of applications and developing a framework to incorporate rescheduling based on performance monitoring of dynamically-changing resources.

The GridLab Resource Management System (GRMS) [46] is an open source meta-scheduling system, developed under the GridLab Project. GRMS is based on dynamic resource selection and discovery, mapping and advanced scheduling methodology, combined with feedback control architecture. With the help of another GridLab Middleware Services, GRMS deals with dynamic Grid environment and resource management challenges, e.g. load balancing among clusters and various workload systems, remote job control or file staging support.

The framework proposed in this thesis also works with similar objective. But in contrast to the previous systems, it employs autonomous agents for performance based resource management of a batch of jobs. This framework has been mapped properly to a multi-agent system (MAS) and a tool `PRAGMA` has been developed. It has been claimed that a realization based on multi-agent system offers advantages in terms of extensibility and flexibility. Well-defined structures to represent job requirements and resource specifications have been used in this framework. Resource brokering technique used in this framework is of *hybrid nature*, which reduces overheads. Moreover, resource selection for multiple concurrent execution of a batch of jobs is performed with the aspiration of *optimizing overall cost / time of resource usage*. The idea is not only to fulfill the requirements of jobs but also to provide cost effective utilization of resources for the entire batch of jobs. This framework can successfully handle sequential as well as parallel jobs. Adaptive execution of jobs in form of rescheduling has been implemented with the help of *mobile agents*, which is a distinctive feature of this framework.

Agent-based framework has been used in other projects as well. The AgentScape project [81, 113] provides a multi-agent infrastructure that can be employed to integrate and coordinate distributed resources in a computational environment. The objective of the AgentScape system is to provide a "minimal but sufficient" environment for agent applications. The A4 [1, 53] project at Warwick has likewise developed a framework for agent based resource management on Grids. As observed by Gradwell P. [45], these multi-agent systems are used to trade for Grid resources at the higher "services" level and not at the base "resource" level. Raw computational resources are normally scheduled using a more classical scheduler, using performance prediction or time based techniques. This work, in contrast to the above mentioned multi-agent systems, focuses on enhancing mainly two aspects of the resource management system (i) resource brokering for a batch of jobs with an objective of maintaining QoS and optimizing the cost of resource usage (ii) improving performance of an application at runtime in case of any violation of the QoS commitment. This multi-agent framework comprises both stationary and mobile agents and

utilizes their ability to provide *resource brokering, regular monitoring of job performance,* and *adaptive execution of concurrent jobs* to achieve preferred high performance.

10.4 Outstanding Issues in Current Work

The results of the experimental evaluation of PRAGMA demonstrate that the multi-agent framework for performance based resource management is capable of achieving high performance and meet the QoS requirement. Nevertheless, the framework has some limitations. This section discusses these limitations and suggests that further work is necessary to resolve these issues.

- This research work mainly deals with independent jobs with no communication requirements. Often the most convenient way to build a complex application is to split the application into subtasks with communication requirements. In this case, some dependent subtasks can get started only after the completion of subtasks on which they depend. Moreover, these subtasks may need to communicate with each other at runtime. To achieve high performance for applications with these types of workflows and dependencies in Grid is a challenging issue and needs to be addressed.
- In the current implementation of PRAGMA, the JRL is being constructed with the help of client provided information. Automatic job modeling, i.e. constructing the JRL automatically from the job itself has not been implemented. An additional module can be implemented which automatically performs job modeling. Thus, the client will submit the job to this module and from the job the module will intelligently generate all kinds of job requirements.
- Time complexity of the algorithm used by this framework for resource selection is $O(N^3)$; the algorithm should be improved in future.
- Cost computation technique used in this work is naïve; this needs to be improved.
- SLA management and negotiation under different administrative domains with distinct sets of policies need to be addressed more carefully and the current implementation of this feature in PRAGMA along with the structure of the SLA needs to be improved.

- Enhanced approach to collect and represent dynamic information on available resource providers at the time of initial scheduling, as well as, if required, before adaptive execution must be explored.
- Current experiments for demonstrating the effectiveness of PRAGMA have been performed on a local Grid test bed. These experiments must be conducted in a more complex and dynamic execution environment.
- Additionally, diverse types of programs (Grid applications) must be tested within the PRAGMA environment.
- Security related aspects are not handled by PRAGMA in this work. The current implementation relies on simple security mechanisms, which need to be improved in future.
- Integrating different local tuning techniques with PRAGMA will enhance the capability of this framework. Detailed performance analysis and performance tuning of Grid applications need to be implemented for this purpose. However, these issues will be addressed in future. Next section presents a guideline for this future work.

10.5 Future Work

Development of the multi-agent-based framework and the PRAGMA tool has suggested several promising avenues for further research. In particular, further work will aim to enhance its performance monitoring and analysing ability. The idea is to make use of multiple Analysis Agents within this multi-agent framework in a hierarchy to monitor the health of the entire execution environment along with the performance of the jobs. A brief overview of this framework is given below.

On top of the execution environment, a hierarchical analysis agent framework will be deployed. Within this framework, agents are divided into the following four logical levels of deployment in descending order: (1) Grid Agent (GA), (2) Grid Site Agent (GSA), (3) Resource Agent (RA) and (4) Node Agent (NA).

These agents interact with each other and perform teamwork to *analyze the performance of large compute-intensive applications*. At the four levels (as shown in Figure 10.1) of agent hierarchy, each agent has some specific responsibilities as discussed below.

At level 1, individual workstations, SMP nodes and each individual node in the clusters are considered as computational resources. *Node Agents (NA)* work on these computational resources and their responsibility is to analyze the performance of the resources and the jobs running on these resources. In case, the execution performance of a job degrades due to a performance problem, which can be solved locally, the NA informs JEM Agent by raising a 'Warning' message immediately. Immediately a Tuning Agent (which is already included within the multi-agent system – see Chapter 4) is invoked for local tuning of the job.

At the next level, a *Resource Agent (RA)* is deployed which analyses the overall performance of the nodes in a cluster. Thus, when a collection of cluster nodes (each node can be an SMP or workstation or a desktop PC) acts as the resource provider, the RA is responsible for looking after the performance at the cluster level (such as cluster level load balancing). Cluster configuration can be homogeneous or heterogeneous. RAs also interact with the JEM Agents and keep them informed about the current status of the resources and execution performance of each job running on the resources.

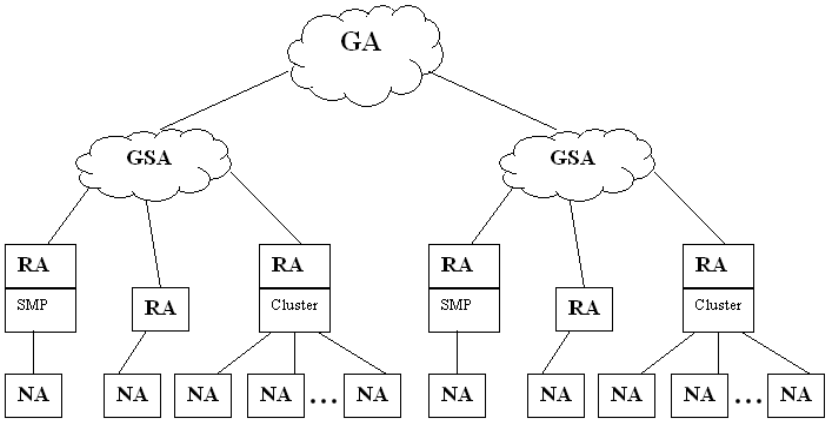


Figure 10.1: The analysis agent hierarchy.

At a higher level, a *Grid Site Agent (GSA)* is deployed. This agent is responsible for the entire Grid resources located at a particular geographical site. The resource providers at a particular Grid site may be under single administrative domain and may have different policies regarding resource availability and usage. The GSA must be able to interact with all these resource providers. A GSA is also useful to identify any fault in any of the resources and also to check whether any of the resources is overloaded. The JCA interacts with the GSAs (at the local site or at a remote site) at the time of migration of a job, in order to find the next suitable resource.

Finally, at the highest level, a *Grid Agent (GA)* is associated with the global Grid and looks after the health of the entire Grid. There is only a single GA, which liaises with all the GSAs and analyzes the overall performance of the resources at each Grid site. This analysis is based on the data collected from the GSAs and the JCAs. Thus each GSA and each JCA must regularly update the GA for enabling it to carry out the analysis.

Future research could also aim at developing a Problem-Solving Environment (PSE) for problem-oriented computing that will provide the entire support to scientific computational problem-solving activities on Grid. A PSE also provides access to diverse range of resources. Further work may be undertaken to design and implement a PSE within `PRAGMA`, which will facilitate problem formulation, algorithm selection, simulation execution, solution visualization etc. A tool enhanced in this way will not only save the effort invested by a Grid user, but will also assist a user to access Grid resources in an efficient manner.

Appendix A

Preliminary Role and Interaction Model

Role Schema: **ResourceBroker (RB)**

Description: This role involves gathering resource information and producing a list of prospective resource providers for each job in the batch. Then selecting the best set of resource providers for the entire batch with the aspiration of overall time or cost optimization, i.e. aiming at minimizing the resource utilization cost (or time) for the entire batch and thereby controlling any kind of over provisioning.

Protocols and Activities:

PrepareJRL, CheckResourceAvailability, MatchJRL, PrepareJobResourceMatrix

Permissions:

Reads	**JDesc, ResourceSpecificationTable**
Generates	**JRL, ResourceProviderList, JobResourceMatrix**

Responsibilities

Liveness:

ResourceBroker=(PrepareJRL, CheckResourceAvailability, MatchJRL)^n.PrepareJobResourceMatrix

Safety:

Number of Jobs=Number of Rows in JobResourceMatrix

Figure A.1: Preliminary role schema of Resource Broker.

Role Schema: **ResourceProvider (RP)**

Description: This role involves preparing ResourceSpecificationMemo, which describes the resources and their quality and also managing the SLAs.

Protocols and Activities:

PrepareResourceSpecification, RegisterResourceProvider, WithdrawResourceProvider, AcceptSLA, RejectSLA, SendResponse

Permissions:

Reads	**SLA**
Generates	**ResourceSpecificationMemo, SLAResponse**
Changes	**ResourceRegistered, ResourceSpecificationTable**

Responsibilities

Liveness:

ResourceProvider=(PrepareResourceSpecification . RegisterResourceProvider) | WithdrawResourceProvider

SLAManager=(AcceptSLA | RejectSLA)$^{+}$. SendResponse

Safety:

-Successful_connection_with_InformationService=true

Figure A.2: Preliminary role schema of Resource Provider.

Role Schema: **JobController (JC)**

Description: This role involves controlling the execution of each of the concurrent jobs (components or tasks of an application) at local level. Besides, maintaining a global view regarding the application's runtime behavior, as well as the functioning of the infrastructure. Also deciding run-time actions for improving the performance whenever an SLA is violated.

Protocols and Activities:

CreateJobMap, PrepareSLA, SetUpSLA, InvokeJobExecutionManager, AwaitWarning, SubmitJob, ProcessJob

Permissions:

Reads	**JobResourceMatrix, JRL, SLA, WarningMessage**
Generates	**SLAs**
Changes	**JobMap**

Responsibilities

Liveness:

JobController = CreateJobMap . (PrepareSLA . SetUpSLA . InvokeJobExecutionManager)^{n}

JobExecutionManager = SubmitJob . ((AwaitWarning)^{w}. Reschedule | InitiateTuning)^{w} || ProcessJob^{w}

Reschedule = (SetUpSLA . SubmitJob)

Safety:

Number of entries in JobMap = Number of SLAs and SLARequests

SLAResponse = true

Figure A.3: Preliminary role schema of JobController.

Role Schema: **Analyzer**

Description: This role involves analyzing the performance of job and environment on the basis of the finalized SLA.

Protocols and Activities:
TransferSLA, MonitorJob, Analyze, RaiseWarning

Permissions:
Reads **SLA**
Generates **WarningMessage**

Responsibilities
Liveness:
Analyzer = TransferSLA. ((MonitorJob . Analyzer)^{w} . RaiseWarning)^{w}
Safety:
-Successful_connection_with_InformationService=true
-Successful_connection_with_MonitoringService=true

Figure A.4: Preliminary role schema of Analyzer.

Role Schema: **PerformanceTuner**

Description: This role involves performance tuning of a job.

Protocols and Activities:
AwaitCall, Tune

Permissions:
Reads **WarningMessage**

Responsibilities
Liveness:
PerformanceTuner = ((AwaitCall)^{w} . Tune)^{w}
Safety:
true

Figure A.5: Preliminary role schema of PerformanceTuner.

CheckResourceAvailability		
RB	**RP**	Input: JRL, RST
Checks whether the resources match a specific JRL.		Output: ResourceProviderList

Figure A.6: Preliminary protocol definition for CheckResourceAvailability.

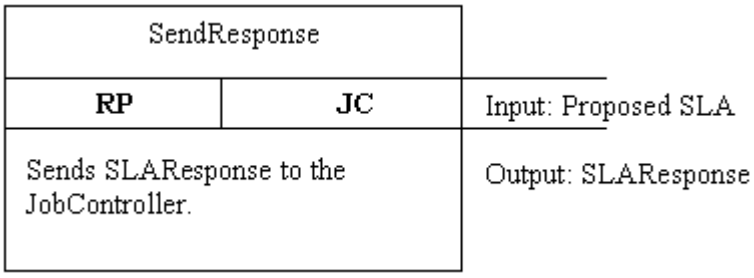


Figure A.7: Preliminary protocol definition for SendResponse.

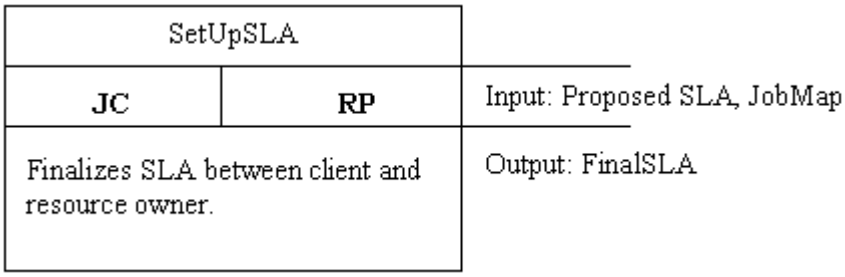


Figure A.8: Preliminary protocol definition for SetUpSLA.

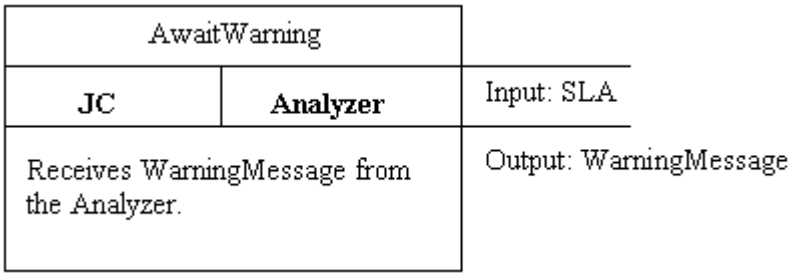


Figure A.9: Preliminary protocol definition for AwaitWarning.

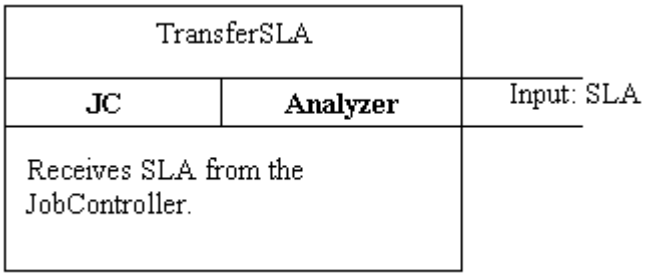


Figure A.10: Preliminary protocol definition for ReceiveSLA.

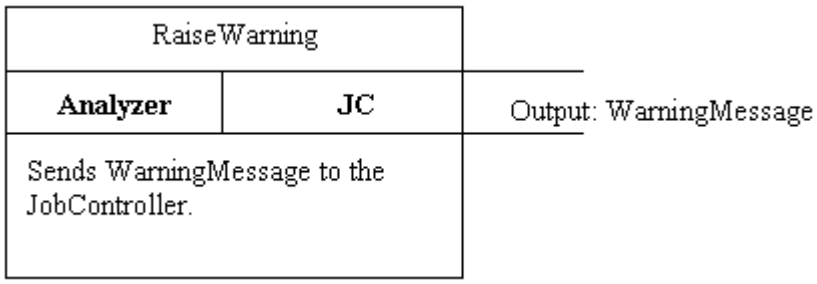


Figure A.11: Preliminary protocol definition for RaiseWarning.

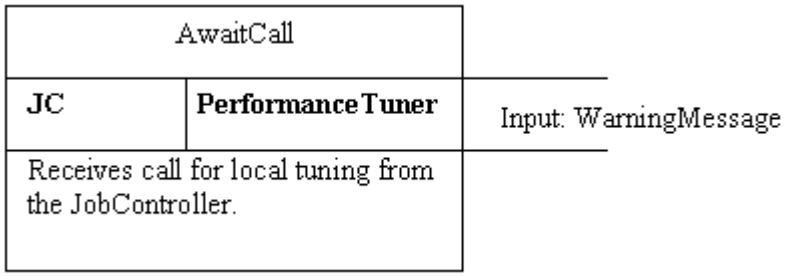


Figure A.12: Preliminary protocol definition for AwaitCall.

Bibliography

[1] A4 Project, web site: http://www.dcs.warwick.ac.uk/research/hpsg/A4/A4.html.

[2] Aglet, web site: http://aglets.sourceforge.net/.

[3] Allan R.J., J.M. Brooke, F. Costen and M. Westhead, "Grid based high Performance Computing", Technical Report of the UKHEC Collaboration, June 2000, web site: http://www.ukhec.ac.uk.

[4] Aydt R. and D. Quesnel compiled "Performance Data Usage Scenarios (Draft 1)", Grid Forum Performance Working Group, October 2000. Available from: http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-3-1.pdf.

[5] Balaton Z., P. Kacsuk, N. Podhorszki and F. Vajda, "Comparison of Representative Grid Monitoring Tools", Technical Report LDS-2/2000, Computer and Automation Research Institute of the Hungarian Academy of Sciences, 2000. Available from: http://www.lpds.sztaki.hu/publications/reports/lpds-2.pdf.

[6] Berman F. and R. Wolski, "The AppLeS project: A status report", In *Proceedings of the 8th NEC Research Symposium*, Berlin, Germany, May 1997. Available from: http://citeseer.ist.psu.edu/berman97apple.html.

[7] Berman F., A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crumme, D. Reed, L. Torczon and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development", In *International Journal of High Performance Computing Applications*, vol.15, no.4, pp. 327-344, November 2001.

[8] Bradshaw M. J., "An Introduction to Software Agents", Software agents, MIT Press, pp. 3-46,1997.

[9] Bull J.M. and M.E. Kambites, ”JOMP-an OpenMP-like Interface for Java”, In *Proceedings of the ACM 2000 conference on Java Grande*, San Francisco, CA, USA, pp. 44-53, June 2000.

[10] Buyya R, H. Stockinger, J. Giddy and D. Abramson, “Economic Models for Management of Resources in Peer-to-Peer and Grid Computing”, In *Proceedings of the SPIE International Conference on Commercial Applications for High-Performance Computing*, Denver, USA, August 2001. Available from: http://www.gridbus.org/papers/economicmodels.pdf.

[11] Buyya R., D. Abramson and J. Giddy, “A Case for Economy Grid Architecture for Service Oriented Grid Computing”, In *Proceedings of the 10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, In conjunction with IPDPS 2001, San Francisco, California, USA, April 2001. Available from: http://www.gridbus.org/papers/ecogrid.pdf.

[12] Buyya R., D. Abramson and J. Giddy, “Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid”, In Proceedings of the 4th International Conference High Performance Computing in Asia-Pacific Region (HPC ASIA’2000), China, pp. 283-289, May 2000.

[13] Byassee J., “Unleash mobile agents using Jini, Leverage Jini to mitigate the complexity of mobile agent applications“, Available from: http://www.javaworld.com/javaworld/jw-06-2002/jw-0628-jini.html.

[14] Cabeza M., M. Clemente and M. Rubio “CacheSim: A Cache Simulator for Teaching Memory Hierarchy Behaviour”, In *Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, Cracow, Poland, pp. 181, September 1999.

[15] Camarinha-Matos L. and H. Afsarmanesh (eds.). “Infrastructure For Virtual Enterprises: Networking Industrial Enterprises”. Kluwer Academic Press: Norwell, 1999.

[16] Castro A., “Designing a Multi-Agent System for Monitoring and Operations Recovery for an Airline Operations Control Centre”, Master-thesis, University of

Porto, Faculty of Engineering, March 2007. Available from: http://antonio.jm.castro.googlepages.com/MasterThesis_AntonioCastro_Final_Rev.pdf.

[17] Chaplin S.J, D. Katramatos, J Karpovich and A.S. Grimshaw, "Resource management in Legion", *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 583-594, October 1999.

[18] Chetty M and R. Buyya, "Weaving Computational Grids: How Analogous Are They with Electrical Grids?", *Computing in Science and Engineering (CiSE)*, USA, vol. 4, no. 4, pp. 61-71, July 2002.

[19] Concordia, web site: http://www.cis.upenn.edu/~bcpierce/courses/629/papers/Concordia-MobileAgentConf.html.

[20] Condor, web site: http://www.cs.wisc.edu/condor.

[21] Condor-G, web site: http://www.cs.wisc.edu/condor/condorg.

[22] Cooper K., A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Melloe-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan and J. Dongarra, "New Grid Scheduling and Rescheduling Methods in the GrADS Project", *International Journal of Parallel Programming*, vol. 33, no. 2, pp. 209-229, June 2005.

[23] Czajkowski K., I. Foster and C. Kesselman, "Resource and Service Management", Chapter 18 of The Grid 2: Blueprint for a New Computing Infrastructure by Ian Foster and Carl Kesselman, Morgan Kaufmann; 2 edition November 2003.

[24] Czakowski K., I. Foster, C. Kesselman, V. Sander and S. Tuecke "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed systems", In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'02)*, Edinburgh-LNCS; pp. 153-183, July 2002.

[25] Dyninst: An Application Program Interface (API) for Runtime Code Generation, web site: http://www.dyninst.org.

[26] European Data Grid Project, web site: http://eu-datagrid.web.cern.ch/eu-datagrid/.

[27] Fangpeng D. and S. G. Akl " Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", Technical Report 2006-504. Available from: http://www.cs.queensu.ca/TechReports/Reports/2006-504.pdf.

[28] Ferreira L., V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan and N. Bieberstein, "Introduction to Grid Computing with Globus", IBM Redbooks, Available from: http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf.

[29] Foster I., and C. Kesselman (eds.). "The Grid: Blueprint for a Future Computing Infrastructure", Morgan Kaufmann: San Francisco, CA, 1998.

[30] Foster I., C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of High Performance Computing Applications*, Sage Publishers, London, UK, vol. 15, no. 3, pp. 200-222, 2001.

[31] Foster I., C. Kesselman, G. Tsudik and S. Tuecke, "A Security Architecture for Computational Grids", In *Proceedings of 5th ACM Conference on Computer & Communication Security*, San Francisco CA USA, pp. 83-92, November 1998.

[32] Foster I., C. Kesselman, J. M. Nick and S. Tuecke, "Grid Services for Distributed System Integration", *IEEE Computer*, vol. 35, no. 6, pp. 37-46, June 2002.

[33] Foster I., C. Kesselman, J. M. Nick and S. Tuecke. "The Philosophy of the Grid: An Open Grid Services Architecture for Distributed systems Integration", Open Grid Services Infrastructure WG, Global Grid Forum, June 2002. Available from: http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[34] Foster I., J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Storey and S. Weerawaranna, "Modeling Stateful Resources with Web Services", Globus Alliance, March 2004. Available from: http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf.

[35] Foster I., K. Czajkowski, D. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling and S. Tuecke, "Modeling and Managing State in Distributed systems: The role of OGSI and WSRF", In *Proceedings of the IEEE*, vol. 93, no. 3, pp. 604-612, March 2005.

[36] Franklin S. and A. Graesser "Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents", In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, Hungary, August 1996. Available from: http://www.msci.memphis.edu/~franklin/AgentProg.html.

[37] Furmento N, A. Mayer, S. McGough, S. Newhouse, T. Field and J. Darlington, "ICENI: Optimisation of Component Applications within a Grid Environment", *Parallel Computing*, vol. 28 no. 12, pp. 1753-1772, December 2002.

[38] Ganglia, web site: http://ganglia.sourceforge.net/.

[39] Gerndt M., R. Wismueller, Z. Balaton, G. Gombas, P. Kacsuk, Z. Nemeth, N. Podhorszki, H. Truong, T. Fahringer, M. Bubak, E. Laure and T. Margalef, "Performance tools for the Grid: State of the Art and Future", Research Report Series, vol. 30, 2004, web site: http://www.fz-juelich.de/apart.

[40] Global Grid Forum, web site: http://www.ggf.org/.

[41] Globus, web site: http://www.globus.org.

[42] Gourlay I., M. Haji, K. Djemame and P. Dew, "Performance Evaluation of a SNAP-Based Grid Resource Broker", Applying Formal Methods: Testing, Performance, and M/E-Commerce, LNCS, vol. 3236/2004, pp. 220-232, September 2004.

[43] GRACE, web site: http://www.grace-ist.org/.

[44] GrADS: Grid Application Development Software Project, web site: http://www.hipersoft.rice.edu/grads/.

[45] Gradwell P., "Overview of Grid Scheduling Systems", Available from: http://peter.gradwell.com/phd/writings/computing-economy-review.pdf.

[46] GridLab Resource Management System, web site: http://www.gridlab.org/grms.

[47] Gt4.0 Manuals, Available from: http://www.globus.org/toolkit/docs/4.0/.

[48] Haji M., K. Djemame and P. Dew "Deployment and Performance Evaluation of a SNAP-based Resource Broker on the White Rose Grid", *Information and Communication Technologies*, ICTTA'06, Damascus, Syria, vol. 2, pp 3365-3370, April 2006.

[49] Hathaway J., "Service Level Agreements: Keeping A Rein on Expectations", In *Proceedings of the 23rd annual ACM SIGUCCS conference on User services (SIGUCCS'95),* ACM Press, St. Louis, Missouri, United States, pp.131-133, October 1995.

[50] Hawkeye, web site: http://www.cs.wisc.edu/condor/hawkeye/.

[51] Hennessy J. L. and D.A. Patterson, "Computer Architecture A Quantitative Approach", Third Edition, Elsevier, 2003.

[52] Horvat D, D. Cvetkovie, V. Milutinovic, P Kooovic and V. Kovacevic, "Mobile Agents and Java Mobile Agents Toolkits", In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, vol. 8, pp. 8029, January 2000.

[53] HPSG, web site: http://www.dcs.warwick.ac.uk/research/hpsg/.

[54] Huedo E, R. S. Montero and I. M. Llorente, "A Framework for Adaptive Execution in Grids", *Software: Practice and Experience*, vol. 34, no. 7, pp. 631-651, June 2004.

[55] James F, T. Tannenbaum, M. Livny, I. T. Foster and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", *Cluster Computing Journal*, Kluwer Academic, vol.5, no. 3, pp. 237-246, July 2002.

[56] Java Grande Forum, web site: http://www.epcc.ed.ac.uk/research/activities/java-grande/.

[57] Jini, web site: http://www.jini.org.

[58] Joseph J, M. Ernest and C. Fellenstein "Evolution of Grid Computing Architecture and Grid Adoption Models", *IBM Systems Journal*, vol. 43, no. 4, pp. 624-645, 2004.

[59] Keller A. and H. Ludwig " Defining and Monitoring Service Level Agreements for Dynamic e-Business", In *Proceedings of the 16th USENIX conference on System administration*, USENIX Association, Philadelphia, PA, pp. 189-204, November 2002.

[60] Kennedy K., M. Mazina, J. M. Mellor-Crummey, K. D. Cooper, L. Torczon, F. Berman, A. A. Chien, H. Dail, O. Sievert, D. Angulo, I. T. Foster, R. A. Aydt, D. A. Reed, D. Gannon, S. L. Johnsson, C. Kesselman, J. Dongarra, S. S. Vadhiyar and R. Wolski, "Toward a Framework for Preparing and Executing Adaptive Grid

Programs", In *Proceedings of the 16th International Parallel and Distributed Processing Symposium Workshop (IPDPS '02)*, IEEE Computer Society Press, pp. 322-326, April 2002.

[61] Krauter K., R. Buyya and M. Maheswaran, " A taxonomy and survey of grid resource management systems for distributed computing", *Software-Practice and Experience*, vol. 32, no. 2, pp. 135-164, February 2002.

[62] Lange D. B. and M. Oshima, "Mobile Agents with Java: The Aglet API", *World Wide Web*, Kluwer Academic, vol. 1, no. 3, pp. 111-121, 1998, Available from: http://zoo.cs.yale.edu/classes/cs434/readings/papers/aglet.pdf.

[63] Legion, web site: http://legion.virginia.edu/.

[64] Li W. and E. B. Allen " An access Control Model for Secure Cluster-Computing Environments", In *Proceedings of the 38th Hawaii International Conference on System Sciences (HISCC '05)*, *- Track 9 - Volume 09*, IEEE Computer Society, pp. 309.2, January 2005.

[65] Maes P., "Artificial Life Meets Entertainment: Life like Autonomous Agents" *Communications of the ACM*, vol. 38, no. 11, pp. 108-114, November 1995.

[66] Maheswaran M, S. Ali, H. J. Siegel, D Hensgen and R. F. Freund "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107-131, 1999.

[67] Mathew L. M., B. N. Chun and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience", *Parallel Computing*, vol. 30, no. 7, July 2004.

[68] Mayes K., G.D. Riley, R.W. Ford, M. Lujan and T.L. Freeman, "The Design of a Performance Steering System for Component-based Grid Applications", *Performance Analysis and Grid Computing*, Kluwer Academic, pp. 111-127, 2004.

[69] Miller B. P, M. D. Callaghan , J. M. Cargille , J. K. Hollingsworth , R. B. Irvin , K. L. Karavanic , K. Kunchithapadam and T. Newhall, "The Paradyn Parallel

Performance Measurement Tool", *IEEE Computer*, vol. 28, no.11, pp. 37-46, November 1995.

[70] Munkres Assignment: http://www.nist.gov/dads/HTML/munkresAssignment.html

[71] Nemeth Z. and V. Sunderam, "A Comparison of Conventional Distributed Computing Environments and Computational Grids", In *Proceedings of the International Conference on Computational Science-Part II (ICCS '02)*, LNCC, vol. 2330, pp. 729-738, April 2002.

[72] Nemeth Z., G. Gombas and Z. Balaton, "Performance Evaluation on Grids: Directions, Issues and Open Problems", In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing PDP'04*, pp. 290-297, February 2004.

[73] NetSolve, web site: http://icl.cs.utk.edu/netsolve/.

[74] Nimrod, web site: http://www.csse.monash.edu.au/~davida/nimrod/.

[75] Nimrod-G, web site: http://www.csse.monash.edu.au/~davida/nimrod/nimrodg.htm.

[76] OASIS Web Service Resource Framework (WSRF): http://www.oasis-open.org.

[77] Odyssey, web site: http://www.genmagic.com/agents/.

[78] OpenMP application program interface, web site: http://www.openmp.org.

[79] OpenSSL, web site: http://www.openssl.org.

[80] Oram A (ed.). "Peer-to-Peer: Harnessing the Power of Disruptive Technologies", O'Reilly Press: U.S.A., 2001.

[81] Overeinder B. J., N. J. E. Wijngaards, M. V. Steen, and F. M. T. Brazier, "Multi-Agent Support for Internet-Scale Grid Management", In *Proceedings of the AISB'02 Symposium on AI and Grid Computing*, London, UK, pp. 18-22, April 2002.

[82] Pablo Research Group, Scalable Performance Tools (Pablo toolkit), web site: http://www-pablo.cs.uiuc.edu.

[83] Padgett J, M. Haji, and K Dijemame " SLA Management in a Service Oriented Architecture", *Computational Science and its Applications-ICCSA 2005*, LNCS, vol. 3483/2005, pp. 1282-1291, May 2005.

[84] Padhorszki N. and P. Kacsuk, "Monitoring Message Passing Applications in the Grid with GRM and R-GMA", *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, LNCS, vol. 2840/2003, pp. 603-610, October 2003.

[85] Padhorszki N. and P. Kacsuk, "Presentation and analysis of Grid Performance Data", *EuroPar-2003 Parallel Processing*, vol. 2790/2004, pp. 119-126, June 2004.

[86] PAPI web site: http://icl.cs.utk.edu/papi/.

[87] Paradyn Parallel Performance Tools, web site: http://www.cs.wisc.edu/paradyn/.

[88] Parashar M. and J. C. Browne "Conceptual and Implementation Model for the Grid", In *Proceedings of the IEEE*, vol. 93, no. 3, pp. 653-668, March 2005.

[89] RealityGrid Project, web site: http://www.realitygrid.org.

[90] Reed D., C. Mendes and S. Whitmore, "GrADS Contract Monitoring", Pablo Group, University of Illinois, Available form: http://www.renci.org/publications/presentations/Grads/GradsMay2002[1].pdf.

[91] Relational Grid Monitoring Architecture, web site: http://www.r-gma.org.

[92] Ribler R.L., H. Simitci and D. Reed, "The Autopilot Performance-Directed Adaptive Control System", *Future Generation Computer Systems*, vol. 18, no. 1, pp 175-187, September 2001.

[93] Rose L. D. and D. Reed, "SvPablo: A Multi-Language Architecture-Independent Performance Analysis System", In *Proceedings of the 1999 International Conference on Parallel Processing*, Fukushima, Japan, pp. 311, September 1999.

[94] Russell S. and P. Norvig, "Artificial Intelligence: A Modern Approach (2^{nd} Edition)", Prentice Hall, December 2002.

[95] SCALEA-G, web site: http://www.par.univie.ac.at/project/scaleag.

[96] Scimark benchmark, web site: http://math.nist.gov/scimark2/index.html.

[97] Shi Z., T. Yu and L. Liu, "MG-QoS: QoS-Based Resource Discovery in Manufacturing Grid", *Grid and Cooperative Computing*, LNCS, vol. 3033/2004, pp. 500-506, April 2004.

[98] Sotomayor B., "The Globus Toolkit 4 Programmer's Tutorial", Available from http://gdp.globus.org/gt4-tutorial/.

[99] Staneva D and P. Gacheva , "Building distributed applications with Java mobile agents", International Workshop NGNT, Available form http://saturn.acad.bg/bis/pdfs/16_doklad.pdf.

[100] Strasser M. and M. Schwehm, "A Performance Model for Mobile Agent Systems", In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, Las Vegas, CA, vol. II, pp. 1132-1140, July 1997.

[101] Sun's Jini Web Site: http://www.sun.com/jini/.

[102] Tierney B, B. Crowley, D. Gunter, M. Holding, J. Lee and M. Thompson, "A Monitoring Sensor Management System for Grid Environments", *Cluster Computing*, vol. 4, no. 1, pp.19-28, March 2001.

[103] Tierney B, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor and R. Wolski, "A Grid Monitoring Architecture", Global Grid Forum, August 2002. Available form: http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf.

[104] Tierney B. and D. Gunter, "NetLogger: a Toolkit for Distributed System Performance Tuning and Debugging", In *Proceedings of the IFIP/IEEE Eighth International Symposium on Integrated Network Management*, Kluwer Academics, vol. 246, pp. 97-100, 2003.

[105] Tomarchio O, L. Vita, and A. Puliafito. "Active Monitoring in GRID Environments using Mobile Agent Technology", In *Proceedings of* 2nd Workshop on Active Middleware Services (AMS'00) in HPDC-9, Pittsburgh, August 2000. Available from: http://www.diit.unict.it/users/otomarch/Papers/ams2000.pdf.

[106] Truong H.L. and T. Fahringer, "SCALEA: A Performance Analysis Tool for Distributed and Parallel Programs", In *Proceedings of the 8th International Euro-Par Conference*, Germany, vol. 2400/2002, pp. 41-55, August 2002.

[107] Truong H.L. and T. Fahringer, "SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid", *Scientific Programming*, vol. 12, no. 4, pp. 225-237, 2004.

[108] Tuecke S., K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, D. Snelling, and P. Vanderbilt, "Open Grid Services Infrastructure (OGSI) Version 1.0", 2003. Available from: http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf.

[109] VGrADS, web site: http://vgrads.rice.edu/.

[110] Voyager, web site: http://elara.tk.informatik.tu-darmstadt.de/LectureNotes/software/voyager/.

[111] Walker R, M. Vetterli, A. Agarwal, R. Sobie and M. Gronager, "A Grid of Grids using Condor-G", Available from: http://hep.phys.sfu.ca/~rwalker/CondorG_chep06.pdf.

[112] White Rose Grid, web site: http://www.wrgrid.org.uk/.

[113] Wijngaards N. J. E, B. J. Overeinder, M. van Steen, and F. M. T. Brazier, "Supporting internet-scale multi-agent systems", *Data Knowledge Engineering*, vol. 41, no. 2-3, pp. 229–245, June 2002. Available from: http://www.iids.org/publications/bnaic2002_dke.pdf.

[114] Wilson A. J., R. Byrom, L. A. Cornwall, M. S. Craig, A. Djaoui, S. M. Fisher, S. Hicks, R. P. Middleton, J. A. Walk, A. Cooke, A. J. G. Gray, W. Nutt, J. Magowan, P. Taylor, J Leake, R. Cordenonsi, N. Podhorszki, B. Coghlan, S. Kenny, O Lyttleton and D. O'Callaghan, "Information and Monitoring Services within a Grid Environment", CHEP'04, Interlaken, Switzerland, September 2004. Available form: http://www.gridpp.ac.uk/papers/chep04_rgma.pdf

[115] Wolski, R, N.T. Spring and J. Hayes, "The Network Weather Service: A distributed performance forecasting service for metacomputing", *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 757-768, 1999.

[116] Wong D, N. Paciorek, and D. Moore, "Java-based mobile agents", *Communications of the ACM*, vol. 42, no. 3, pp. 92-102, March 1999.

[117] Wooldridge M, N. R. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", In *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 3, no.3, pp. 285-312, 2000.

[118] Xu C., "Naplet: A Flexible Mobile Agent Framework for Network-Centric Applications", In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS'02)*, pp 219-226, April 2002.

[119] Yang C, P. Shih and K. Li "A High-Performance Computational Resource Broker for Grid Computing Environments", In *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05)*, pp 333-336, March 2005.

[120] Zambonelli F, N. R. Jennings and M. Wooldridge. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering Methodology*, vol. 12, no. 3, pp. 317-370, July 2003.